Technical manual for FLBEIA a R package to conduct Bio-Economic Impact assessments using FLR (version 1.15)

Dorleta García, Raúl Prellezo, Sonia Sánchez, Marga Andrés, Agurtzane Urtizberea & Itsaso Carmona

June 30, 2017

Abstract

FLBEIA (FL Bio-Economic Impact Assessment) is an R package build on top of FLR libraries. The purpose of the package is to provide a flexible and generic simulation model, also called FLBEIA, to conduct Bio-Economic Impact Assessments of harvest control rule based on management strategies under a Management Strategy Evaluation (MSE) framework. The model is divided into two main blocks, the operating model (OM) and the management procedure model (MPM). The OM is formed by the biological, the fleet and the covariates components and the MPM by the observation, the assessment and the management advice components.The model is multistock, multifleet and seasonal, and uncertainty is introduced by means of montecarlo simulation. The algorithm has been coded in a modular way to ease its checking and to make it flexible. The package provides functions that describe the dynamics of the different model components and the user chooses which of the functions are used in each specific case study. Furthermore, for some of the components, if the functions provided within FLBEIA do not fulfill the requirements of a specific case study, the user can code the functions that describe better the dynamics of those components. Therefore, due to the wide choice of functionality and flexibility that provides the model, we can define it as a framework more than as a model. Main limitations of the model are that the stocks must be age structured or aggregated in biomass (length structure is not allowed), and that spatial dimension is not considered explicitly. However, spatial characteristics could be modeled assigning stocks and/or fleets/metiers to specific areas.

Contents

1 Introduction

The idea of `FLBEIA` comes from the similarities between the different models developed to perform bio-economic analysis in AZTI-Tecnalia. These models were pieces of code re-written in order to match with the specific case study or fishery. These pieces, in many cases, reflected exactly the same processes with similar dynamics that had to be slightly adapted to the different case studies. Therefore, in order to ease the job of the modelers, we decided to develop not a model but a framework in which a model is built. This model can be constructed combining already existing functions or developing new functions and combining them with existing ones. The choice of the kind of model to be used in a specific case study depends on the questions asked, which implies that not any model can be considered valid for all purposes.

Big advances have been done the last years in the field of bio-economic modelling with the development of models such as, Fishrent (**?**), Fcube (**?**), FcubeEcon (**?**) among others, and with the development of also some theoretical and partial assessments. However, until now there is no an universal model that can be applied to address all fishery management issues. Thus, we developed `FLBEIA` with the objective to integrate many of the models available in a common bio-economic impact assessment framework as a package of FLR (**?**) in (**?**). `FLR` (**?**) was built with the goal of developing a common framework to facilitate collaboration within and across disciplines (e.g. biological, ecological, statistical, mathematical, economic, and social) and, in particular, to ensure that new modelling methods and software are more easily validated and evaluated, and more widely available once developed.

The package `FLBEIA` contains the model called `FLBEIA`, a collection of functions and new `S4` classes developed to facilitate the simulation of fishery systems in response to different types of management strategies. The model allows the evaluation of different management strategies, in a wide variety of case studies and scenarios, under Management Strategy Evaluation framework (**?????**), and identifies the potential economic and biological consequences of a proposed policy action.

The main characteristics of `FLBEIA` package are:

- It is coded in a generic, flexible and extensible way.
- Provides functions to condition the simulations, to run them and to analyze the results.

In fact, a mayor effort has been set on the second functionality, namely the simulation model.
The main characteristics of the `FLBEIA` simulation model are:

- The model is fully biological-economic coupled and provides fully integrated bio-economic assessment.
- The model deals with multi-species, multi-fleet and multi-*metier* situations.
- The model can be run using seasonal steps (smaller or equal to one year).
- It is generic, flexible and extensible.
- Uncertainty can be introduced in almost any of the parameters used.

A conceptual diagram of the model is shown in Figure **??**. The simulation is divided in two main blocks: the Operating Model (OM) and the Management Procedure Model (MPM). The OM is the part of the model that simulates the real dynamics of the fishery system and the MPM is the part of the model that simulates the whole management process.

The OM has three components that can interact among themselves:

1. The biological populations or stocks.
2. The fleets.
3. The covariates. They can be of any nature; environmental, economical or technical.

The MPM has also three components:

1. The data collected from the OM.
2. The observed population obtained through the application of a set of assessment models to the observed data.
3. The management advice obtained from the application of harvest control rules (HCR) to the observed populations.

The model is built modularly with a top-down structure that has, at least, four levels:

1. In the first level (top level), there is only one function, `FLBEIA` function. It calls the functions on the second level in a determined order and it links the main components (stocks, fleets, covariates, data, observed population and management advice) of the OM and MPM.

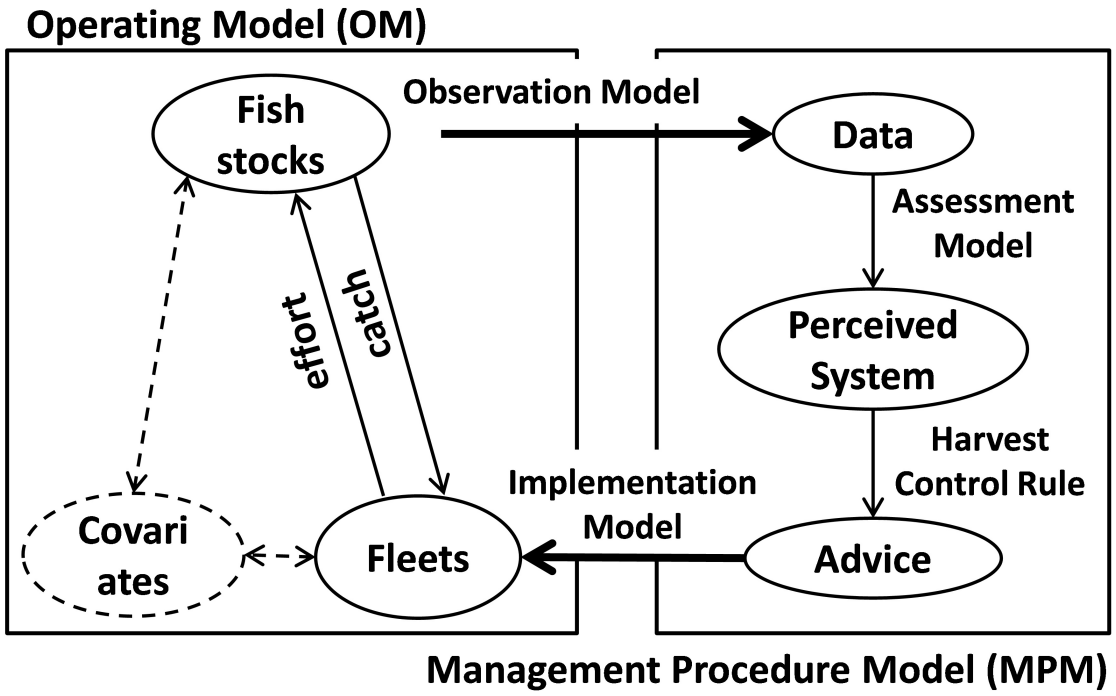

Figure 1: Conceptual representation of the main components modelled in FLBEIA. Source: (**?**)

2. The functions in the second level correspond with the generation of each of the components in the Figure **??**. The OM components project the objects one season forward: `biols.om` projects the stocks, `fleets.om` projects the fleets and `covars.om` projects the covariates. The MPM components generate the objects necessary to produce the management advice, they generate the objects based on OM objects and they operate at most once a year: `observation.mp` generates the data, `assessment.mp` generates the observed population and `advice.mp` generates the management advice. They take the input objects and return only those related to the component they belong to.
3. The functions in the third level define the specific dynamics of each component and they are chosen by the user in each simulation. They are always called by a second level function and in some cases, a third level function also calls fourth level functions. For example a function that describes the dynamics of an age structured population can call a stock recruitment function. In this way, a function used to describe age structured populations can be combined with different stock recruitment relationships.
4. The functions in the fourth level are called by functions in the third level and are used to model the most basic processes in the simulation. They are coded as a function and selected by the user because it could be interesting to use the same third level function together with different fourth level functions, as in the case of age structured population and stock recruitment functions.

This top down structure allows avoiding the classical structure of separated biological and economic (and social) modules (that could be integrated or not). Therefore, when the model is designed and the modeler takes the decision of including a particular characteristic, it does not make any difference if the characteristic is biological or economical, only matters at which level the characteristic is.

`FLBEIA` framework permits to incorporate new third and lower level functions or to modify them, while first and second level ones are fixed. Changing first or second level functions would imply a different approach, but the existing third and lower level functions would be useful.

In the next Sections `FLBEIA`'s conceptual model and its specifications are explained. Firstly, in Section **??**, the conceptual model characterizes the main components as well as the feedbacks and loops among them. Secondly, in Section **??**, it is explained how to run FLBEIA to perform MSE. Thirdly, in Section **??**, the model specification describes the components, the currently available functions by level, and how to use them within the `FLBEIA` package. Next, in Section **??**, a way to easily condition the model is presented. And finally, in Section **??**, the `FLBEIA` function output is described.

2 The concept of FLBEIA

The simulation model is divided in two main blocks, the Operating Model (OM) and the Management Procedure Model (MPM). This division is part of the requirements of the MSE approach, that is, the model includes a mathematical representations of the *real world* (OM), the *observed world* (MPM) and the interactions between them.

2.1 Operating model

The OM is the part of the model that simulates the real dynamics of the fishery system. It is divided in three components or operating models, the biological, the fleets and the covariates operating model. It runs in seasonal time steps, and it projects the components in each time step. Firstly, it updates the biological component, secondly the fleet component and finally the covariates component.

Biological component

The biological component simulates the population dynamics of the stocks. The number of populations is, in principle, unlimited. The limitation could come from memory problems with R and/or the operating system. The stocks can be described as age structured populations or as biomass dynamics populations, since length structured populations models are not supported by the simulation algorithm. Each stock can follow a different population dynamics model and is projected independently. It does not mean that they cannot be interdependent between them but the order in which these biological components are updated has to be decided and it will affect the results obtained.

Fleet component

The fleet component simulates the behaviour and dynamics of the individual fleets. As the number of the stocks, the number of fleets is in principle unlimited. The limitation could come from memory problems with R and/or the operating system used. The activity of the fleets is divided into metiers. The metiers are formed by trips that have the same catchability for all the stocks. Fleet fishing effort and effort share among metiers are independently updated for each fleet in each season. Fleet catchability and/or capacity is updated annually, independently for each fleet, through capital dynamics according to its own economic performance.

Covariates component

This part of the model incorporates all the variables that are not part of the biological or fleet components and that affect any of the operating model components or the management process. The number of covariates is, in principle, unlimited. The limitation could come from memory problems with R and/or the operating system used.

Links among and within components

The links within the OM components are not restricted by the general settings of the simulation model. Therefore, it is the user who decides which are the links that should be included in the model. The possible links that can be included are:

- The link within the *biological* component, where catch affects abundance.
- The link within the *fleets* component, where fleet capacity affects fishing effort.
- The link between the *biological and fleets* components, where fishing effort and fish abundance affects catches.

2.2 Management procedure model

The Management Procedure Model (MPM) is divided into 3 components: the observation, the assessment and the management advice. The observation component produces the required data to run the assessment. Then, the assessment component is applied to those data to obtain the observed populations. Finally, the management advice component produces a management advice based on the observed populations. MPM procedure is applied yearly in the appropriate season of the year. Not necessarily in the last season, for example, it can be simulated as in the case of anchovy in the Bay of Biscay, where management is applied from the mid-season of one year to the mid-season of the next year. Simulations with multi-annual advice is also possible.

Observation component

The observation component generates the required objects to run the assessments. Three types of objects can be generated:

- Stocks.
- Fleets.
- Abundance indices.

Stocks and abundance indices objects are generated independently, stock by stock, whereas fleets are observed jointly. These objects are generated based on the variation that is introduced in the components of the OM. This variation can be due to:

- Introducing uncertainty to the OM variables, or
- adjusting the OM variables to the assessment model requirements which is going to be used in the next step (e.g. collapsing the dimensions -age, season,...), or
- adjusting the OM variables to the legal conditionings (TACs, quotas, TAE, discards,...).

Assessment component

Assessment models are applied on a stock by stock basis and they can vary from stock to stock.

Management advice component

The management advice component produces a set of indicators (determined by the user) useful for policy making. The management advice is produced based on the output obtained from the observation and assessment components. The advice is first applied at single stock level and after that it can be applied at fleet level.

3 Running FLBEIA

3.1 Input objects

FLBEIA requires some input arguments to run a simulation. There are two types of arguments: the main arguments, which give information on the stocks, the fleets and the covariates, and the control arguments, which control the behaviour of the main and the second level functions (see Section **??**). The main arguments contain biological information on the stocks (biols, SRs and BDs), information on the fleets (fleets), additional variables (covars) and information on management (indices, advice). Regarding the control arguments, there is a control object related to the main function FLBEIA (main.ctrl), whereas the others relate to the main arguments (biols.ctrl, fleets.ctrl, covars.ctrl, obs.ctrl, assess.ctrl, advice.ctrl). For detailed information on the input objects required see Section **??**.

In order to easy the creation of the objects with the appropriate object format some additional functions has been implemented to create the inputs (see Section **??**). Additionally, several examples has been coded for guidance (see Section **??**).

3.2 Main function: FLBEIA

To perform biological and economic simulations it is necessary to invoke the main function, FLBEIA, which calls to different subfunctions to perform the simulations depending on the control elements set.
FLBEIA function is called as follows:

```
FLBEIA(biols, SRs, BDs, fleets, covars, indices, advice, main.ctrl, biols.ctrl,
        fleets.ctrl, covars.ctrl, obs.ctrl, assess.ctrl, advice.ctrl)
```

For more details on the FLBEIA function see Section **??**.

3.3 Output object

The output of FLBEIA function is a list containing information on the expected evolution of the fish stocks (biols), the fleets (fleets), the covariates (covars), the provided advice (advice), the assessed stocks and observed indices (stocks and indices), the control elements for the fleets (fleets.ctrl) and the versions of the different packages used to run the simulations (pkgs.versions). All the elements except stocks and pkgs.versions correspond with the updated versions of the objects used in the call to FLBEIA.
It has the following structure:

```
        list(biols='FLBiols', fleets='FLFleetsExt', covars='FLQuants',
    advice='list(TAC,TAE,quota.share)', stocks='FLStocks', indices='FLIndices',
                    fleets.ctrl='list', pkg.versions='matrix')
```

Description of the outputs:

`biols:` `FLBiols` object containing historical and future "real" evolution of the stock.

`fleets:` `FLFleetsExt` object containing historical and future "real" evolution of the fleets regarding effort exerted, distribution among metiers, catches, prices and so on.

`covars:` Named list containing historical and future evolution of the different covariates.

`advice:` List containing information on management advice (e.g. TAC,TAE,quota.share).

`stocks:` Named list with one element per stock of class `FLStock` or `NULL`, if a `FLStock` is not needed to run the assessment. Contains the perceived stocks used in the management procedure to produce the management advice. For details on `FLStock` object see Figure **??**.

`indices:` Named list with one element per stock of class `FLIndices` or `NULL`, if a `FLIndices` is not needed to run the assessment.

`fleets.ctrl:` Control object used for the `fleets.om` function.

`pkgs.versions:` Matrix indicating the packages and package version used along the simulation.

4 FLBEIA functions

4.1 First level function: FLBEIA

`FLBEIA` function is a multistock, multifleet and seasonal simulation algorithm coded in a generic, flexible and extensible way. It is generic because it can be applied to any case study that fit into the model restrictions. The algorithm is made up by third and fourth level functions specified by the user. In addition of the existing functions new ones can be defined and used if necessary. This is why we define the model as flexible and extensible.

To determine the simulation, the third- and fourth-level functions must be specified in the main function `FLBEIA`. For this purpose it has a control argument associated to each second level function. These control arguments are lists which include the name of the functions to be used in the simulations and any extra argument required by those functions that is not already contained in the main arguments. `FLBEIA` function is called as follows:

```
FLBEIA(biols, SRs, BDs, fleets, covars, indices, advice, main.ctrl, biols.ctrl,
         fleets.ctrl, covars.ctrl, obs.ctrl, assess.ctrl, advice.ctrl)
```

Main arguments:

`biols:` An `FLBiols` object (list of `FLBiol` objects). The object must be named and the names must be the same as in the `SRs` object, the `BDs` object and the `catches` slots within `FLFleetExts` object. For details on `FLBiol` object see Figure **??**.

`SRs:` A list of `FLSRsim` objects. This object is a simulation version of the original `FLSR` object. The object must be named and the names must be the same as in the `FLBiols` object. For details on `FLSRsim` object see Figure **??**.

`BDs:` A list of `FLBDsim` objects. This object is similar to `FLSRs` object but oriented to simulate population growth in biomass dynamics populations. The object must be named and the names must coincide with those used in `FLBiols` object. For details about `FLBDsim` object see Figure **??**.

`fleets:` An `FLFleetsExt` object (list of `FLFleetExt` objects). `FLFleetExt` object is almost equal to the original `FLFleet` object but the `FLCatch` object in `catch` slot has been replaced by `FLCatchExt` object. The difference between `FLCatch` and `FLCatchExt` objects is that `FLCatchExt` has two extra slots `alpha` and `beta` used to store Cobb-Douglas production function parameters, α and β, (**??**). α corresponds with the exponent of effort and β to the exponent of biomass. The `FLFleetsExt` object must be named and these names must be consistently used in the rest of the arguments. For details about `FLFleetExt` object see Figure **??**.

`covars:` An `FLQuants` object. This object is not used in the most basic configuration of the algorithm. Its content depends on the third or lower level functions that make use of it.

`indices:` A list of `FLIndex` objects. Each element in the list corresponds with one stock. The list must be named and the names must be the same as in the `FLBiols` object. For details about `FLIndex` object see Figure **??**.

`advice`: A list. The class and content of its elements depends on two functions, the function in `fleet.om` defined to simulate fleets' effort and the function used to produce advice in `advice.mp`.

Control arguments:

`main.ctrl`: Controls the behaviour of the main function, `FLBEIA`. For details on `main.ctrl` object see Table **??**.

`biols.ctrl`: Controls the behaviour of the second level function `biols.om`. For details on `biols.ctrl` object see Table **??**.

`fleets.ctrl`: Controls the behaviour of the second level function `fleets.om`. For details on `fleets.ctrl` object see Table **??**.

`covars.ctrl`: Controls the behaviour of the second level function `covars.om`. For details on `covars.ctrl` object see Table **??**.

`obs.ctrl`: Controls the behaviour of the second level function `observation.mp`. For details on `obs.ctrl` object see Table **??**.

`assess.ctrl`: Controls the behaviour of the second level function `assessment.mp`. For details on `assess.ctrl` object see Table **??**.

`advice.ctrl`: Controls the behaviour of the second level function `advice.mp`. For details on `advice.ctrl` object see Table **??**.

4.2 Second level functions

4.2.1 Biological component: `biols.om`

The call to the function within `FLBEIA` is done as:

```
biols.om(biols, fleets, SRs, BDs, covars, biols.ctrl, year, season)
```

This function projects the stocks one season forward. The projection is done independently stock by stock by the third level function specified for each stock in `biols.ctrl` object. Currently, there are three population dynamics functions implemented, one corresponding to age structured populations, `ASPG`, the second one to biomass dynamics populations, `BDPG` and another one to fixed populations (given as input), `fixedPopulation`. These functions do not include predation among stocks, but this kind of models could be implemented and used in the algorithm if necessary.
Control arguments:

`biols.control`: This argument is a list which contains the necessary information to run the third level functions that are called by `biols.om`. The elements depend on the third and lower level functions used to describe the dynamics of the stocks. The list must contain at least one element per stock and the name of the element must coincide exactly with the name used in `biols` argument so it can be used to link the population with its dynamics model. At the same time, each of these elements must be a list with at least one element, `growth.model`, which specifies the name of the function used to describe population dynamics (options: `ASPG`, `BDPG` or `fixedPopulation`).

For example:

```
> biols.ctrl

$NHKE
$NHKE$growth.model
[1] "ASPG"

$CMON
$CMON$growth.model
[1] "BDPG"

$FAKE
$FAKE$growth.model
[1] "ASPG"
```

4.2.2 Fleets component: `fleets.om`

The call to `fleets.om` function within `FLBEIA` is done as:

```
fleets.om(fleets, biols, covars, advice, fleets.ctrl, advice.ctrl, year, season)
```

This function projects the fleets one season forward. The main argument, `fleets`, is an object of class `FLFleetsExt` (for more detail see Section **??**)

The function is divided in three processes related to fleet dynamics: the effort model, the price model and the capital model. Effort and capital models are fleet specific, whereas price model is fleet and stock specific. First, `fleets.om` calls the effort model and it updates the slots related to effort and catch. The effort models are called independently fleet by fleet. Then, `fleets.om` calls the price model in fleet by fleet and stock by stock basis, which updates the `price` slot in the `fleets` object. Finally, but only in the last season of the year, the function calls the capital model. Thus, investment and disinvestment is only done annually. The capital model is called independently fleet by fleet.

Effort model: This part of the model simulates the tactical behaviour of the fleet every season and iteration. In each time step and iteration, the effort exerted by each individual fleet and its effort-share among metiers is calculated depending on the stock abundance, management restrictions or others. After that, the catch produced by the combination of effort and effort-share is calculated and `discards, discards.n, landings, landings.n` slots are filled. Other stored variables in `fleets.ctrl` could also be updated here, for example `quota.share`, as a result of the exerted effort.

The effort model is specified at fleet level, so each fleet can follow a different effort model. At the moment there are 4 functions available: `fixedEffort, SMFB, SSFB` and `MaxProfit`. To write new functions for effort, it must be taken into account that the input arguments must be found among `fleets.om` function arguments and that the output must be a list with updated `FLFleetsExt` and `fleets.ctrl` objects, i.e.:

```
list(fleets = my_fleets_obj, fleets.ctrl = my_fleets.ctrl_obj)
```

Price Model: The price model updates the price-at-age at stock, metier and fleet level in each time step and iteration.

At the moment, there are 2 functions available: `fixedPrice` and `elasticPrice`. To write new functions for price it must be taken into account that the input arguments must be found among `fleets.om` function arguments and that the output must be a list with an updated `FLFleetsExt` object.

Capital Model: This module is intended to simulate the strategic behaviour of the fleets, namely, the investment and disinvestment dynamics. The model is applied at fleet level and in an annual basis and can affect fleets' capacity and catchability. Catchability could be modified through investment in technological improvement and capacity as a result of an increase (investment) or decrease (disinvestment) in the number of vessels. Changes in fleets' capacities could produce a variation in quota share among fleets, for example. Thus, the corresponding change would have to be done in `fleets.ctrl` object.

At the moment, there are 2 functions available: `fixedCapital` and `SCD`. To write new functions for capital dynamics, as for effort and price, it must be taken into account that the input arguments must be found among `fleets.om` function arguments and that the output must be a list with updated `FLFleetsExt` and `fleets.ctrl` objects.

Control arguments:

`fleets.ctrl:` The most simple example of fleet dynamics model and hence the most simple `fleets.ctrl` object correspond with the model where all the parameters in `fleets` object are given as input and maintained fixed within the simulation. This is obtained using the third level functions, `fixedEffort, fixedPrice` and `fixedCapital` which do not need any extra arguments. In the case of two fleets, `FL1` and `FL2`, where `FL1` catches 3 stocks, `ST1, ST2` and `ST3` and `FL2` catches `ST1` and `ST3` stocks, the `fleets.ctrl` could be created using the following code:

```
>fleets.ctrl <- list()

# The fleets
>fleets.ctrl[['FL1']] <- list()
>fleets.ctrl[['FL2']] <- list()

# Effort model per fleet.
>fleets.ctrl[['FL1']]$effort.model <- 'fixedEffort'
>fleets.ctrl[['FL2']]$effort.model <- 'fixedEffort'

# Price model per fleet and stock.
>fleets.ctrl[['FL1']][['ST1']]$price.model  <- 'fixedPrice'
>fleets.ctrl[['FL1']][['ST2']]$price.model  <- 'fixedPrice'
```

```
>fleets.ctrl[['FL1']][['ST3']]$price.model  <- 'fixedPrice'

>fleets.ctrl[['FL2']][['ST1']]$price.model  <- 'fixedPrice'
>fleets.ctrl[['FL2']][['ST3']]$price.model  <- 'fixedPrice'

# Capital model by fleet.
>fleets.ctrl[['FL1']]$capital.model  <- 'fixedCapital'
>fleets.ctrl[['FL2']]$capital.model  <- 'fixedCapital'

> fleets.ctrl
  $FL1
  $FL1$effort.model
  [1] "fixedEffort"

  $FL1$ST1
  $FL1$ST1$price.model
  [1] "fixedPrice"

  $FL1$ST2
  $FL1$ST2$price.model
  [1] "fixedPrice"

  $FL1$ST3
  $FL1$ST3$price.model
  [1] "fixedPrice"

  $FL1$capital.model
  [1] "fixedCapital"

  $FL2
  $FL2$effort.model
  [1] "fixedEffort"

  $FL2$ST1
  $FL2$ST1$price.model
  [1] "fixedPrice"

  $FL2$ST3
  $FL2$ST3$price.model
  [1] "fixedPrice"

  $FL2$capital.model
  [1] "fixedCapital
```

4.2.3 Covariates component: `covars.om`

`covars.om` projects `covars` object one season forward. `covars` object is a named list and the class and dimension of each element will depend on the function used to project it into the simulation.
The call to `covars.om` function within `FLBEIA` is done as:

```
covars.om(biols, fleets, covars, advice, covars.ctrl, year, season)
```

Internally, for each element in the `covars` list, it calls to the third level functions specified in the `covars.ctrl` object. At the moment, there exist 2 third level functions: `fixedCovar`, which is used to work with variables that are input parameters not updated within the simulation and `ssb.get`, which is used to get the real Spawning Stock Biomass of one of the simulated stocks.

The economic variables used in the capital dynamics model `SCD` should be stored in `covars` object and updated in each step using values in `fleets` object.
Control arguments:

`covars.ctrl:` This argument is a named list with one element per covariate and the names of the list must match those used to name the `covars` object. Each of the elements is, at the same time, a

list with, at least, one element, `dyn.model`, which defines the dynamics of the covariate in question (options: `fixedCovar`, `ssb.get`).

This way of working could be useful, for example, for environmental variables such as sea surface temperature that could affect catchability or recruitment in the fleet and biological operating models respectively and that are external to fishery system.

A covariate with a non-trivial dynamics could be the abundance of certain animal which is not commercially exploited by the fleet, but which abundance affects the natural mortality of any of the exploited stocks. In this case, 2 extra functions will be needed, the function that defines the dynamics of the covariate and the function that models the natural mortality of the stock as a function of the abundance of the animal. The first function should be declared in `covars.ctrl` argument and the former one in `biols.ctrl` argument as a stock dynamics model.

4.2.4 Observation component: `observation.mp`

The observation component generates the necessary data to run the assessment models. The main function is `observation.mp` and it calls third level functions which generate 3 possible objects, a `FLStock`, a `FLIndices` or a `FLFleetsExt` object. The `FLStock` and `FLIndices` objects are generated independently for each stock and the `FLFleetsExt` object jointly for all the fleets.
The call to `observation.mp` function within `FLBEIA` is done, stock by stock, as follows:

```
observation.mp(biols, fleets, covars, indices, advice, obs.ctrl, year, season, stknm)
```

where `stknm` is the name of the stock to be observed and its name matches with those used in the `biols` object.

The output of `observation.mp` is a list with 3 elements. The first element, `stock` is an object of class `FLStock` or `NULL`, if a `FLStock` is not needed to run the assessment. The second element, `indices`, is a named list with one element per stock and its names correspond with those used in `biols` object. The elements of the `indices` list are of class `FLIndices` or `NULL`, if a `FLIndices` is not needed to run the assessment. The third element, `fleets.obs`, is an observed version of the original `fleets` object. At the moment, there is no third level function implemented to generate observed fleets.

As the management process is currently run in a yearly basis, the `unit` and `season` dimensions are collapsed in all the observed objects. Moreover, if the management process is being conducted at the end of year `y` the observed objects extend up to year `y-1`, whereas they extend up to year `y` in the cases when management process is conducted in any other season as it happens in reality.
Control arguments:

`obs.ctrl`: The `obs.ctrl` object must be a named list where the names used correspond with those used in the `FLBiols` object. Each stock element is, at the same time, a list with two elements (`stkObs` and `indObs`) and these two elements are once again lists. A scheme of `obs.ctrl` object is presented in Table **??**.

The `stkObs` element is a list with the arguments necessary to run the third level function used to generate the `FLStock` object. In the list there must be at least one element, `stkObs.model`, with the name of the third level function that will be used two generate the `FLStock` object. If it is not required to generate a `FLStock` object, then `NoObsStock` value should be assigned to `stkObs.model` argument and this function will return the `NULL` object.

The `indObs` element is a list with one element per index in the `FLIndices` object. Each element of the list is, at the same time, a list with the arguments necessary to run the third level function used to generate the `FLIndex` object. In the list there must be at least one element, `indObs.model`, with the name of the third level function that will be used two generate the `FLIndex` object. If it is not required to generate a `FLIndices` object, then `indObs` element will be set equal to `NoObsIndex` instead of a list and this will return the `NULL` object instead of a `FLIndices` for the corresponding stock.

4.2.5 Assessment component: `assessment.mp`

The assessment component applies an existing assessment model to the stock data objects generated by the observation model (`FLStock` and `FLIndices`). The assessment models are applied stock by stock, independently.
The call to `assessment.mp` function within `FLBEIA` is done as follows:

```
assessment.mp(stocks, fleets.obs, indices, assess.ctrl, datayr, stknm)
```

where `stknm` is the name of the stock to be assessed and its name corresponds with one of those used in the `biols` object.

The output of the function is a list of `FLStocks` with `harvest`, `stock.n` and `stock` slots updated. Within `FLBEIA` no new assessment models are provided, but the models already available in `FLR` can be used.

Control arguments:

`assess.ctrl:` This argument is a named list with one element per stock, where the names must coincide with those used in the `biols` object. The elements must have at least one element, `assess.model`, which defines the name of the assessment model to be used for each stock. Furthermore, if the assessment model to be used is non-trivial (i.e. different to `NoAssessment`), the list must contain a second argument `control` with the adequate control object to run the assessment model.

4.2.6 Management advice component: `advice.mp`

The management advice component generates an advice based on the output of assessment and/or observation components.

The call to `advice.mp` function within `FLBEIA` is done, stock by stock, as follows:

```
advice.mp(stocks, fleets.obs, indices, covars, advice, advice.ctrl, year, season, stknm)
```

where `stknm` is the name of the stock to be assessed and its name correspond with one of those used in the `biols` object.

The output of the function is an updated `advice` object.

Depending on the structure of the third level functions used to generate advice and to simulate fleet dynamics, the advice could be an input advice (effort, temporal closures, spatial closures -implicitly through changes in catchability-...) or an output advice (catch).

`advice:` The structure of `advice` object is open and it is completely dependent on the third level functions used to describe fleet dynamics and to generate the advice. For example, if `SMFB` and `annualTAC` are used to describe fleet dynamics and generate the advice respectively, then `advice` is a list with two elements, `TAC` and `quota.share`. `TAC` is an annual `FLQuant` with the `quant` dimension used to store stock specific TACs and, `quota.share` is a named list with one element per stock being the elements `FLQuant`-s with `quant` dimension used to store fleet specific annual quota share.

Control arguments:

`advice.ctrl:` This argument is a named list with one element per stock and one more element for each fleet. The names must coincide with those used to name `biols` object and the name of the extra argument must be `fleets`. The elements of the list are, at the same time, lists with at least one element, `HCR.model`, with the name of the model used to generate the single stock and fleet advice depending on the case.

4.3 Third level functions

4.3.1 Population growth functions

The following population growth functions are currently defined:

`fixedPopulation`: Fixed population function

In this function all the parameters are given as input, because there is not any population dynamics simulated. For the stocks for which we select its dynamics as fixed population, natural mortality (i.e. `biols[[stock.name]]@m`) has to be set equal to 0 and additionally, if the population is aggregated in biomass, biomass growth (i.e. `BDs[[stock.name]]@gB`) has also to be set equal to 0.

`ASPG`: Age Structured Population Growth function

The function `ASPG` describes the evolution of an age structured population using an exponential survival equation for existing age classes and a stock-recruitment relationship to generate the recruitment. The recruitment can occur in one or more seasons. However, the age is measured in integer years and the seasonal cohorts are tracked separately. The seasonal cohorts and their corresponding parameters are stored in the '`unit (u)`' dimension of the `FLQuant`-s. And all the individuals move from one age group to the following one in the 1st of January. Thus, being ϕ the recruitment function, RI the reproductive index, N the number of individuals, M the natural mortality, C the catch, a_0 the age at recruitment, s_0 the season when the recruitment was spawn, and a, y, u, s the subscripts for age, year, unit and season respectively, the population dynamics can be written mathematically as:

If $s = 1$,

$$N_{a,y,u,1} = \begin{cases} \phi\left(RI_{y=y-a_0,s=s-s_0}\right) & , a = a_0 \\ (N_{i_a} \cdot e^{-\frac{M_{i_a}}{2}} - C_{i_a}) \cdot e^{-\frac{M_{i_a}}{2}} & , a_0 < a < A \\ (N_{i_{A-1}} \cdot e^{-\frac{M_{i_{A-1}}}{2}} - C_{i_{A-1}}) \cdot e^{-\frac{M_{i_{A-1}}}{2}} + & \\ (N_{i_A} \cdot e^{-\frac{M_{i_A}}{2}} - C_{i_A}) \cdot e^{-\frac{M_{i_A}}{2}} & , a = A \end{cases} \tag{1}$$

where $i_a = (a-1, y-1, u, ns)$, $i_{A-1} = (A-1, y-1, u, ns)$ and $i_A = (A, y-1, u, ns)$.

If $s > 1$,

$$N_{a,y,u,s} = \begin{cases} \phi\left(RI_{y=y-a_0,s=s-s_0}\right) & , a = a_0 \\ (N_{i_a} \cdot e^{-\frac{M_{i_a}}{2}} - C_{i_a}) \cdot e^{-\frac{M_{i_a}}{2}} & , a_0 < a < A \end{cases} \tag{2}$$

where $i_a = (a, y, u, s-1)$.

And the reproductive index RI is given by:

$$RI_{y-a_0,s} = \sum_a \sum_u (N \cdot wt \cdot mat \cdot fec \cdot exp - (M \cdot M_{spwn} + F \cdot F_{spwn}))_{a,y-a_0,u,s} \tag{3}$$

where wt is the mean weight, mat is the percentage of mature individuals, fec is the fecundity parameter, M_{spwn} and F_{spwn} are the proportion of natural and fishing mortality, respectively, occurring before spawning.

The stock-recruitment relationship ϕ is specified in the `model` slot of corresponding `FLSRsim` object. `FLSRsim` object enables modeling a great variety of stock-recruitment relationships depending on its functional form and seasonal dynamics. Details on available stock-recruitment relationships are given in Section **??**.

`BDPG`: Biomass Dynamics Population Growth function

The function `BDPG` describes the evolution of a biomass dynamics population, i.e. a population with no age, stage or length structure. The population is aggregated in biomass, B, and the growth of the population, g is a function of the current biomass and the catch C. The model is mathematically described in Equation **??**:

$$B_{s,y} = \begin{cases} B_{s-1,y} + g(B_{s-1,y}) - C_{s-1,y} & , s \neq 1 \\ B_{ns,y-1} + g(B_{ns,y-1}) - C_{ns,y-1} & , s = 1 \end{cases} \tag{4}$$

where s and y are the subscripts for age and year, respectively, and ns is the number of seasons. As `FLBEIA` is seasonal, the equation also depends on the season. The growth model g and its parameters are specified, respectively, in the `model` and `params` slot of corresponding `FLBDsim` class. Currently only Pella and Tomlinson model (**?**) is implemented to model growth, but new models can be defined if needed.

The following parameterization of the growth model has been implemented:

$$g(B) = B \cdot \frac{r}{p} \cdot \left[1 - \left(\frac{B}{K}\right)^p\right] \tag{5}$$

where r is the intrinsic rate of population increase, K the carrying capacity and p the assymetry parameter. Additionally, there has been added a restriction in order to avoid negative values. This arises when population is at high biomass values (well above carrying capacity) and outside the range of observed biomass levels in the past. Moreover, it doesn't occur for larger catches that result in lower biomass levels. Intuitively, this seemed to be contradictory because the population collapsed in the absence of catches case and remained stable for higher catch levels. Therefore, in the absence of catches, we restrict the biomass to be α times the carrying capacity ($B_t \leq \alpha \cdot K$). In other words, $B_t = min(B_t, \alpha \cdot K)$. But note that:

$$\alpha \geq 1$$
$$\alpha \leq \left(\frac{p}{r} + 1\right)^{\frac{1}{p}}$$

Note that when we introduce stochasticity in the parameters of the Pella-Tomlinson model (e.g. from a Bayesian model or from a bootstrapping) we have a range of values for the surplus production model. So that α must be smaller than the minimum value across iterations: $\alpha \leq min_i((p_i/r_i + 1)^{1/p_i})$. The value of α has to be introduced by the user in the `BDs[[stock.name]]@alpha`. Above restrictions will be checked in `FLBEIA` and it will print an error if they are not fullfilled. Finally, note that these restrictions arise from the no catch case. So, even after restricting the biomass, there might be cases when some levels of catches lead to negative biomasses.

When working with populations structured in biomass, the biomass values has to be stored in the `*.n` slots, whereas `*.wt` slots has to be set to 1.

4.3.2 Effort models

The following effort model functions are currently defined:

`fixedEffort`: Fixed effort model

In this function all the parameters are given as input except discards and landings (total and at age). The only task of this function is to update the discards and landings (total and at age) according to the catch production function specified in `fleets.ctrl` argument.

Two arguments need to be declared as elements of `fleets.ctrl` if this function is used, `effort.model = 'fixedEffort'` and `catch.model`. The last argument is used to specify the catch production function that will be used to generate the catch. Note that first argument must be declared at fleet level (i.e `fleets.ctrl[[fleet.name]]$effort.model`), second argument at fleet and stock level (i.e. `fleets.ctrl[[fleet.name]][[stock.name]]$catch.model`) and that catch production model corresponds with a fourth level function. For more details see Section **??**.

`SMFB`: Simple Mixed Fisheries Behaviour model

This model is a simplified version of the behavior of fleets that work in a mixed fisheries framework. The function is seasonal and assumes that effort share among metiers is given as input parameter.

In each season, the effort of each fleet, f, is restricted by the seasonal landing quotas or catch quotas of the stocks that are caught by the fleet. Additionaly, the option of Landing Obligation (LO) is included. The following steps are followed in the calculation of effort:

1. Compare the overall seasonal quotas, $\sum_f Q_{f,s,st} \cdot TAC$, with the abundances of the stocks. If the ratio between overall quota and abundance exceeds the seasonal catch threshold, $\gamma_{s,st}$, reduce the quota share in the same degree. Mathematically:

$$Q'_{f,s,st} = \begin{cases} Q_{f,s,st} & \text{, if } \frac{\sum_f Q_{f,s,st} \cdot TAC}{B_{s,st}} \leq \gamma_{s,st} \\ Q_{f,s,st} \cdot \frac{B_{s,st} \cdot \gamma_{s,st}}{\sum_f Q_{f,s,st} \cdot TAC} & \text{, otherwise} \end{cases} \tag{6}$$

2. According to the catch production function, calculate the efforts corresponding to the landing or catch quotas, $Q'_{f,s,st} \cdot TAC$, of the individual stocks, $\{E_{f,s,st_1}, \ldots, E_{f,s,st_n}\}$.

3. Based on the efforts calculated in the previous step, calculate an unique effort, $E_{f,s}$. To calculate this effort the following options can be used:

   `max:` The maximum among possible efforts, $\hat{E}_{f,s} = \max_{j=1,\ldots,n} E_{f,s,st_j}$

   `min:` The minimum among possible efforts, $\hat{E}_{f,s} = \min_{j=1,\ldots,n} E_{f,s,st_j}$

   `mean:` The mean of possible efforts, $\hat{E}_{f,s} = \text{mean}_{j=1,\ldots,n} E_{f,s,st_j}$

   `previous:` The effort selected is the effort most similar to previous year effort on that season,

$$\hat{E}_{f,s} = \left\{ E_{f,s,st} : \left| 1 - \frac{E_{f,s,st}}{E_{f,y-1,s}} \right| = \min_{j=1,\ldots,n} \left| 1 - \frac{E_{f,s,st_j}}{E_{f,y-1,s}} \right| \right\}$$

   `stock.name:` The effort corresponding to `stock.name` is selected: $\hat{E}_{f,s} = E_{f,s,\texttt{stock.name}}$

   If there is LO, instead of using the option chosen by the user, the option to calculate the unique effort will be the minimum among possible efforts.

4. When LO is applied, calculate the new effort using the exemptions and flexibilities de Minimis, year tranfer and quota swap.

   - *de Minimis*: The fleet is allowed to discard a percentage of the quota to increase the effort in order to catch other stocks.
   - *year transfer*: The fleet can borrow next year's quota to catch it in the current year.
   - *quota swap*: A percentage of the quota of one stock can be transfered to the effort limiting stock if two stocks are in the same group. These groups are specifiyed by the user.

5. Compare the effort, $\hat{E}_{f,s}$, with the capacity of the fleet, κ_f (capacity must be measured in the same units as effort and it must be stored in the `capacity` slot of the `FLFLeetsExt` object). If the capacity is bigger, then the final effort is unchanged and if the capacity is smaller, the effort is set equal to the capacity, i.e.:

$$E_{f,s} = \begin{cases} \kappa_f & \text{, if } \kappa < \hat{E}_{f,s} \\ \hat{E}_{f,s} & \text{, if } \kappa \geq \hat{E}_{f,s} \end{cases} \tag{7}$$

6. The catch corresponding to the effort selected is calculated for each stock and compared with the corresponding quota. If the catch is not equal to the quota and the season is not the last one, the seasonal quota shares of the rest of the seasons are reduced or increased proportionally to their weight in the total share. The shares are changed in such a way that the resultant annual quota share is equal to the original one. In case the difference between actual catch and that corresponding to the quota exceeds the quota left over in the rest of the seasons, the quota in the rest of the seasons is canceled. Mathematically for season i where $s \leq i \leq ns'$:

$$Q''_{f,i,st} = \max\left(0, Q'_{f,i,st} + (Q'_{f,s,st} - Q''_{f,s,st}) \cdot \frac{Q'_{f,i,st}}{\sum_{j>s} Q'_{f,j,st}}\right) \tag{8}$$

where Q' denotes the quota share obtained in the first step and Q'' the new quota share.

The `fleets.ctrl` argument in `SMFB` function

`SMFB` function requires several control arguments at global and fleet level that are described below.

Global arguments:

`catch.threshold:` This element is used to store $\gamma_{s,st}$ parameter described in the first step of `SMFB` function algorithm. The element must be a `FLQuant` object with dimension `[stock = nstk, year = ny, unit = 1, season = ns, area = 1, iter = ni]`, where the names in the first dimension must match with those used to name `FLBiols` object. Thus, the thresholds may vary between stocks, seasons, years and iterations. The elements of the object are proportions between 0 and 1 that indicate the maximum percentage of the stock that can be caught in each season. The reason to use this argument is that it is reasonable to think that it is impossible to fish all the fish in the sea. Thus, although the TAC is very large the actual catch will be restricted to $\gamma_{s,st} \cdot B_{s,st}$.

`seasonal.share:` A named `FLQuants` object, one per stock, with the proportion of the fleets' TAC share that 'belongs' to each season, so the sum along seasons for each fleet, year and iteration should be equal to 1. The elements must be `FLQuant` objects with dimension `[fleet = nf, year = ny, unit = 1, season = ns, area = 1, iter = ni]`, where the names in the first dimension must match with those used to name `FLFleetsExt` object. The names of the `FLQuants` must match stock names used in the `FLBiols` object.

Fleet level arguments (i.e. `fleets.ctrl[[fleet.name]]`):

`effort.model: 'SMFB'`.

`effort.restr:` alternative values are `'max'`, `'min'`, `'mean'`, `'previous'` or `'stock.name'` (the name of one of the stocks caught by the fleet).

`max:` The fleet will continue fishing until the catch quotas of all the stocks are exhausted.

`min:` The fleet will stop fishing when the catch quota of any of the stocks is exhausted.

`previous:` Among the efforts obtained under each stock restriction the effort most similar to the previous year effort will be selected.

`stock:` The fleet will continue fishing until the catch quota of '`stock`' is exhausted. (This could correspond, for example, with a situation where the catch of one stock is highly controlled.)

These options are explained mathematically above when the `SMFB` function is described step by step. There are two alternatives: one option for all years or one option for each year (vector with the length ny).

`restriction:` Alternative values are `'catch'` or `'landings'`. Assigned value depends on wether the efforts are calculated according to catch or landings restriction. There are two alternatives: one option for all years or one option for each year (vector with the length ny).

`LandObl:` Logical or vector with a logic value for each year. If it is `TRUE` for that year, LO rule is applied. The fleet has to stop fishing when they reach the first quota of the stocks. Therefore, the unique effort will be the minimum among possible efforts.

`LandObl_minimis:` Vector with a logic value for each year. If it is `TRUE` for that year, *de Minimis* exemption is used.

`LandObl_yearTransfer:` Vector with a logic value for each year. If it is `TRUE` for that year, *year Transfer* flexibility is used.

`LandObl_minimis_p:` Matrix with values between 0 and 1, the maximum percentage of quota that the fleet could increase for each stock in each year.

`LandObl_yearTransfer_p:` Matrix with values between 0 and 1, the maximum percentage of quota of each stock that the fleet could borrow from next year's quota.

`LandObl_discount_yrTranfer:` Matrix with values between 0 and 1. The discount to be applied if in the previous year was used that amount. This object is used to store the percentage used from the next year's quota.

`LO_stk_grp:` named vector with length equal to the number of stocks, same number for the same group of stocks to swap the quotas.

Fleet/stock level arguments (i.e. `fleets.ctrl[[fleet.name]][[stock.name]]`):

`catch.model:` The name of the fourth level function which gives the catch production given effort and biomass (aggregated or at age). The function must be coherent with `SMFB` and the function used to simulate the population growth. At the moment, two functions are available `CobbDouglasAge` and `CobbDouglasBio`. For more details see Section **??**.

`SSFB`: Simple Sequential Fisheries Behaviour model

Simple sequential fisheries behaviour is related to those fleets whose fishing profile changes with the season of the year. `SSFB` function models the behaviour of fleets that work in a sequential fisheries framework. It is assumded that, in each season, the fleet, f, has only one target species or stock, st, thus the metier, m, is defined on the basis of the season and target species, resulting only in one target species per each metier.

In each season, s, the effort allocated to each species, st, or metier, m, follows the historical trend (in order to capture the seasonality of each species fishing season), but it is restricted to the remaining catch quota of the fleet.

Therefore, production function is applied at metier level, but the production has some restrictions, in both catches, C, and effort, E, that are described through the following steps:

1. Calculate the total quota that corresponds to each fleet, CQ, from the historical data and estimate remaining quota for the fleet, $RQ_{s,f,st}$, deducting the catches from previous seasons.

$$RQ_{s,f,st} = CQ_{f,st} - \sum_{ss<s} C_{ss,f,st} = TAC \cdot QS_{f,st} - \sum_{ss<s} C_{ss,f,st}$$

   Where QS is the quota share and C the catches.

2. Compare the total remaining quotas with the abundances of the stocks. If the ratio between remaining quotas and abundance exceeds the seasonal catch thershold, $\gamma_{s,st}$, then reduce the remaining quota the same amount.

$$RQ'_{s,f,st} = \begin{cases} RQ_{s,f,st} & \text{, if } \frac{\sum_f Q_{s,f,st}}{Bs,st} \leq \gamma_{s,st}; \\ RQ_{s,f,st} \cdot \frac{Bst,s \cdot \gamma_{s,st}}{\sum_f RQ_{s,f,st}} & \text{, otherwise.} \end{cases}$$

3. Initially expected effort, $\hat{E}_{s,f}$, is shared between different metiers (i.e. species) month by month on the basis of historical seasonal effort pattern.

$$\hat{E}_{s,m,st} = \hat{E}_{s,f} \cdot E_{s,m} = \kappa_f \cdot PED_{s,f} \cdot E_{s,m}$$

   Where $E_{s,m}$ is the effort share by metier, $PED_{s,f}$ is the percentage of effective days and κ_f is the fleet's capacity.

4. Expected catches ,$\hat{C}_{s,m,st}$, corresponding to that initial effort, are calculated through the Cobb-Douglas catch production function at metier and stock level, seasonally.

5. If the expected catches resulting from the previous step are higher than the remaining quota corresponding to each metier (Step 2), there is extra effort which has to be reallocated among the other species.

$$\text{If } \hat{C}_{s,f,st} > RQ_{s,f,st} \Rightarrow \hat{C}_{s,f,st} = RQ_{s,f,st} \Rightarrow E_{s,m,st} < \hat{E}_{s,m,st};$$
$$\text{else } \hat{C}_{s,f,st} \leq RQ_{s,f,st} \Rightarrow \hat{C}_{s,f,st} = C_{s,f,st} \Rightarrow E_{s,m,st} = \hat{E}_{s,m,st}.$$

6. The reallocation of remaining effort, $\hat{E}_{s,m,st} - E_{s,m,st}$, can be performed in different ways:
   - Proportionally to the price and availability of the species in a given season, or
   - proportionally to the effort allocated to the remaining metiers.
7. This is repeated stock by stock until no effort remains to be allocated or all the TACs are exhausted

The `advice` argument in `SSFB` function

`SSFB` function requires arguments in the `advice` object as described below.

Global arguments:

`quota.share`: A named `FLQuants` object, one per stock, with the total proportion of TAC that 'belongs' to each fleet each year and dimension `[fleet = nf, year = ny, unit = 1, season = 1, area = 1, iter = ni]`. The '`fleet`' dimension names must match fleets' names. And the `FLQuants` must match stock names. For each year and iteration the sum of the proportions must be equal to 1.

The `fleets.ctrl` argument in `SMFB` function

`SSFB` function requires several control arguments at global and fleet level that are described below.

Global arguments:

`catch.threshold:` A `FLQuant` object with dimension `[stock = nst, year = ny, unit = 1, season = ns, area = 1, iter = ni]`, which contains the proportion of biomass that total catch of stock cannot exceed, i.e. the previously mentioned $\gamma_{s,st}$ parameter.

Fleet level arguments (i.e. `fleets.ctrl[[fleet.name]]`):

`effort.model:` '`SSFB`'

`restriction:` '`catch`'. Related to quota threshold.

`effectiveDay.perc:` A `FLQuant` object with dimension `[quant = 1, year = ny, unit = 1, season = ns, area = 1, iter = ni]`, which contains the proportion of days expected to be effective in a season (i.e. in which the fleet will go out fishing), the previously mentioned PED parameter (see Step 3).

`effort.realloc:` Alternative values are `NULL` or '`curr.eff`'. Element used to describe how does the remaining effort have to be reallocated between the rest of the metiers targeting stocks for which there is already remaining quota.

`NULL:` The same proportion is assigned for all metiers.

`curr.eff:` Effort is reallocated proportionally to the expected effort share.

Fleet/stock level arguments (i.e. `fleets.ctrl[[fleet.name]][[stock.name]]`):

`TAC.OS.model:` Function to model the TAC overshoot. Currently the only available function is `TAC.OS.triangCond`, which simulates a triangular distribution funcion for TAC overshoot, in the range `(min, max)` and a peak in the `mode`.

`TAC.OS.triangCond.params:` A named numeric vector of dimension 3. Corresponding to the parameters required by `TAC.OS.triangCond` function, `min`, `max` and `mode`.

`discard.TAC.OS:` Logical. If `TRUE`, the TAC overshoot is discarded, in other case the TAC overshoot is incorporated to landings.

`MaxProfit`: Maximization of profit under a TAC constraint model

This second model used to simulate *mixed fisheries* dynamics calculates the total effort and the effort allocation among metiers that maximises the profit of the fleet. The total effort is constrained by the capacity of the fleet (capacity unit has to be converted in the same unit as effort) and by the catch quota of some of the stocks. Mathematically:

$$\max_{E_f, \gamma_{f,1}, \ldots, \gamma_{f,n_{MT,f}}} \sum_m \sum_{st} \sum_a L_{st,a,f,m} \cdot P_{st,a,f,m} - E_f \cdot \gamma_{f,m} \cdot VaC_{f,m} - FxC_f \cdot n_{V_f} \tag{9}$$

with the constraints:

$$\begin{cases} 0 \leq \gamma_{f,m} \leq 1 \text{ and } \sum_m \gamma_{f,m} = 1 \\ E_f \leq \kappa_f, \\ C_{st,f} \leq QS_{st,f} \quad \text{for } st \in \Delta_f. \end{cases} \tag{10}$$

where P is the price of the fish landed, VaC the variable cost of fishing effort, which depends on the metier and is given as cost per unit of effort, FxC the fixed costs of each fishing unit, which is given at fleet level and in terms of cost per vessel, n_V is the number of vessels in the fleet, κ is the capacity, defined as the maximum effort that the fleet can execute in each season, QS is fleet's TAC share and Δ is the set of stocks for which the constraint must be fulfilled. In biomass dynamic populations, landings and prices are given at stock level.

The `fleets.ctrl` argument in `MaxProfit` function

`MaxProfit` function requires several control arguments at fleet level that are described below.

Fleet level arguments (i.e. `fleets.ctrl[[fleet.name]]`):

`stk.cnst:` A character with the name of the stock that +++++

Fleet/stock level arguments (i.e. `fleets.ctrl[[fleet.name]][[stock.name]]`):

`TAC.OS.model:` Function to model the TAC overshoot. Currently the only available function is `TAC.OS.triangCond`, which simulates a triangular distribution funcion for TAC overshoot, in the range `(min, max)` and a peak in the `mode`.

`TAC.OS.triangCond.params:` A named numeric vector of dimension 3. Corresponding to the parameters required by `TAC.OS.triangCond` function, `min`, `max` and `mode`.

`discard.TAC.OS:` Logical. If `TRUE`, the TAC overshoot is discarded, in other case the TAC overshoot is incorporated to landings.

`MaxProfitSeq`: Maximization of profit under a TAC constraint model for a sequential fishery

+++++ MARGA/AGURTZANE: describir las diferencias con respecto a MaxProfit

4.3.3 Price models

The following price model functions are currently available:

`fixedPrice`: Fixed price model

The prices are given as input data and are unchanged within the simulation. Only the function name, `fixedPrice`, must be specified in `price.model` element in `fleets.ctrl` object.

```
fleets.ctrl[[fleet.name]][[stock.name]]$price.model <- 'FixedPrice'
```

`elasticPrice`: Elastic price model

This function implements the price function used in **?**:

$$P_{a,y,s,f} = P_{a,0,s,f} \cdot \left(\frac{L_{a,0,s,f}}{L_{a,y,s,f}} \right)^{e_{a,s,f}} \tag{11}$$

It uses base price, $P_{a,0,s,f}$, and base landings, $L_{a,0,s,f}$ to calculate the new price $P_{a,y,s,f}$ using a elasticity parameter $e_{a,s,f}, (e \geq 0)$. If the base landings are bigger than current landings the price is increased and decreased if the contrary occurs. a, y, s and f correspond to the subscripts for age, year, season and fleet, respectively. For simplicity, the iteration subscripts have been omitted but all the elements in the equation are iteration dependent. As prices could also depend on total landings instead of on fleet's landings, there is an option to use $L_{a,0,s}$ instead of $L_{a,0,s,f}$ in the formula above.

Although price is stored at metier and stock level in `FLFleetsExt`, this function assumes that price is common to all metiers within a fleet and it is calculated at fleet level.

The `fleets.ctrl` argument in `fixedPrice` function

When `elasticPrice` is used, the following arguments must be specified, at fleet and stock level (i.e. for `fleets.ctrl[[fleet.name]][[stock.name]]`):

`price.model: 'fixedPrice'`.

`pd.Pa0:` An array with dimension `[age = na, season = ns, iter = ni]` to store base price, $P_{a,0,s,f}$.

`pd.La0:` An array with dimension `[age = na, season = ns, iter = ni]` to store base landings, $L_{a,0,s,f}$.

`pd.els:` An array with dimension `[age = na, season = ns, iter = ni]` to store price elasticity, $e_{a,s,f}$.

`pd.total:` Logical. If `TRUE` the price is calculated using total landings and if `FALSE` the landings of the fleet in question are used to estimate the price.

4.3.4 Capital models

The following capital model functions are currently available:

`fixedCapital`: Fixed capital model

The capacity and catchability are given as input data and are unchanged within the simulation. Only the function name, `fixedCapital`, must be specified in `capital.model` element of `fleets.ctrl` object.

```
fleets.ctrl[[fleet.name]]$capital.model <- 'FixedCapital'
```

`SCD`: Simple Capital Dynamics model

In this simple function catchability is not updated, it is an input parameter, and only capacity is updated depending on some economic indicators. The following variables and indicators are defined at fleet and year level (fleet and year subscripts are omitted for simplicity):

FuC: Fuel Cost.

CrC: Crew Cost.

VaC: Variable Costs.

FxC: Fixed Costs (repair, maintenance and other).

CaC: Capital Costs (depreciation and interest payment).

Rev: Revenue, given by the formula:

$$Rev_f = \sum_m \sum_s \sum_a L_{m,s,a} \cdot P_{a,s}$$

where L is the total landings, P the price and m, s, a the subscripts for metier, season and age, respectively.

BER: Break Even Revenue, the revenues that make profit equal to 0.

$$BER = \frac{FxC + CaC}{1 - \frac{FuC}{Rev} - \frac{CrC}{Rev - FuC} + \frac{FuC \cdot CrC}{Rev \cdot (Rev - FuC)} - \frac{VaC}{Rev}}$$

In principle the investment, Inv, is determined by:

$$Inv_0 = \frac{Rev - BER}{Rev}$$

But not all the profits are dedicated to increase the fleet, thus:

$$Inv = \eta \cdot \frac{Rev - BER}{Rev}$$

where η is the proportion of the profits that is used to buy new vessels. Furthermore, investment in new vessels will only occur if the operational days of existing vessels is equal to maximum days. If this occurs, the investment/disinvestment decision, Ω, will follow the rule below:

$$\Omega_y = \begin{cases} Inv & \text{, if } (Inv_0 < 0 \text{ and } \eta \cdot |Inv_0| < \omega_1) \mid (Inv_0 > 0 \text{ and } \eta \cdot |Inv_0| < \omega_2) \\ -\omega_1 * \kappa_{y-1} & \text{, if } Inv_0 < 0 \text{ and } \eta \cdot |Inv_0| > \omega_1 \\ \omega_2 * \kappa_{y-1} & \text{, if } Inv_0 > 0 \text{ and } \eta \cdot |Inv_0| > \omega_2 \end{cases} \quad (12)$$

where ω_2 stands for the limit on the increase of the fleet relative to the previous year and ω_1 for the limit on the decrease of the fleet relative to the previous year.

4.3.5 Covariates models

The following covariates model functions are currently available:

`fixedCovar`: Fixed covariates model

The covariates that follow this model are given as input data and are unchanged within the simulation. Only the function name, `fixedCovar`, must be specified in `process.model` element of `covars.ctrl` object.

```
covars.ctrl[[covar.name]]$process.model <- 'fixedCovar'
```

`ssb.get`: model to get the SSB of one stock

This function is used for including the real Spawning Stock Biomass of one of the simulated stocks as a covariate when fitting the stock recruitment relationship of another stock. In the `covars.ctrl` object the following elements need to be specified:

`process.model: 'ssb.get'`.

`ssb.stock:` Character string with the name of the stock for which you want to get the SSB.

`spwn.sson:` Numeric argument with the spawning season of this stock.

`sr.covar:` Character string with the name of the stock for wich you want to include the influence of stock `ssb.stock` in its stock recruitment relationship.

4.3.6 Observation models: catch and biological parameters

The functions in this section are used to generate a `FLStock` object from `FLBiol` and `FLFleetsExt` objects. The former is used to fill the slots relative to biology, (`stock.wt`, `mat` and `m` slots), and the last to fill the slots relative to catch, landings and discards. Whereas `harvest, stock` and `stock.n` slots are left empty and `harvest.spwn` and `m.spwn` are set equal to 0.

`age2ageDat` This function creates an age structured `FLStock` from age structured `FLBiol` and `FLFleetsExt` objects. The slots of the `FLStock` object are filled in the following way:

`landings.n:` Observed landings at age are obtained from `fleets` object, summing them up along seasons, units, metiers and fleets. After summing up, two sources of uncertainty are introduced, one related to aging error and the second one related to misreporting. Aging error is specified through `ages.error` argument, an array with dimension `[age = na, age = na, year = ny, iter = ni]`. For each year and iteration, each element `(i,j)` in the first 2 dimensions indicates the proportion of individuals of age `i` that are wrongly assigned to age `j`, thus the sum of the elements along the first dimension must be equal to 1. For each year and iteration, the real landings at age are multiplied matricially with the corresponding sub-matrix of `ages.error` object. Afterwards, the second source of uncertainty is introduced multiplying the obtained landings at age by `land.nage.error`, an `FLQuant` with dimension `[age = na, year = ny, unit = 1, season = 1, area = 1, iter = ni]`. Once uncertainty is introduced in landings at age and weight at age, the total landings are computed and compared with the TAC. If landings are lower than $\texttt{TAC} \cdot \texttt{TAC.ovrsht}$, the observed landings at age are unchanged, but if they were higher, the landings at age would be reduced by $\frac{1}{\texttt{TAC.ovrsht}}$ where `TAC.ovrsht` is a positive real number.

`landings.wt:` Observed landings weight at age is derived from `fleets` object, averaging it along seasons, units, metiers and fleets. After averaging, 2 sources of uncertainty are introduced, one related to aging error and the second one related to misreporting. Aging error is the same as the one used in the landings at age. For each year and iteration, the real weight at age is weighted by the proportion of landings in each age group and multiplied matricially with the corresponding submatrix of `ages.error` object. Afterwards, the second source of uncertainty is introduced multiplying the obtained weight at age by `land.wgt.error` an `FLQuant` with dimension `[age = na, year = ny, unit = 1, season = 1, area = 1, iter = ni]`.

`discards.n:` Observed discards at age are obtained in the same way as the landings but summing up the discards instead of landings and using, in the second source of error, the object `disc.nage.error`, an `FLQuant` with dimension `[age = na, year = ny, unit = 1, season = 1, area = 1, iter = ni]`. The object `ages.error` is the same as the one used in the derivation of landings at age.

`discards.wt:` Observed discards weight at age is obtained in the same way as the landings but averaging along discards weight instead of landings weight and using, in the second source of error, the object `disc.wgt.error`, an `FLQuant` with dimension `[age = na, year = ny, unit = 1, season = 1, area = 1, iter = ni]`. The object `ages.error` is the same as the one used in the derivation of landings at age.

`discards, landings:` Observed total discards and landings are derived from observed landings and discards at age and their corresponding weight.

`catch, catch.n, catch.wt:` Slots related to observed catches are derived from the observed landings and discards at age and their corresponding weight.

`m:` Observed natural mortality at age is obtained from `m` slot in `FLBiol`. Adittionally 2 sources of uncertainty are introduced. Firstly, for each year and iteration, this mortality is matriciallly multiplied by the ageing error (the same as the one used for catch related slots). Afterwards, the object is multiplied by `nmort.error`, where `nmort.error` is an `FLQuant` with dimension `[age = na, year = ny, unit = 1, season = 1, area = 1, iter = ni]`. `nmort.error` is used to introduce multiplicative uncertainty in the observation of natural mortality.

`mat:` Observed proportion of individuals mature at age is obtained from `mat` slot in `FLBiol` object. Firstly, for each year and iteration, this proportion is matriciallly multiplied by the ageing error (the same as the one used for catch related slots). Afterwards, the object is multiplied by `mat.error`, where `mat.error` is an `FLQuant` with dimension `[age = na, year = ny, unit = 1, season = 1, area = 1, iter = ni]`. `mat.error` is used to introduce multiplicative uncertainty in the observation of maturity.

`bio2bioDat` This function creates a `FLStock` object aggregated in biomass from `FLBiol` and `FLFleetsExt` objects aggregated in biomass.

`m, mat, landings.n, landings.wt, discards.n, discards.wt, catch.n, catch.wt` : Observed values for these slots are set to `NA`.

`discards:` Observed discards are obtained as follows: the discards are summed up along fleets and metiers and then uncertainty (observation error) is introduced using a multiplicative error. This multiplicative error is specified through `disc.bio.error` argument an `FLQuant` with dimension `[quant = 1, year = ny, unit = 1, season = 1, area = 1, iter = ni]`.

`landings:` Observed landings are derived in the same way as discards but the argument used to introduce uncertainty is called `land.bio.error` in this case. Once uncertainty is introduced in landings, they are compared with the TAC. If the landings are lower than $\texttt{TAC} \cdot \texttt{TAC.ovrsht}$, the observed landings are unchanged but if there were higher the landings would be reduced by $\frac{1}{\texttt{TAC.ovrsht}}$, where `TAC.ovrsht` is a positive real number.

`catch:` Observed catch slot is equal to the sum of landings and discards.

`age2bioDat` This function creates a `FLStock` aggregated in biomass from age structured `FLBiol` and `FLFleetsExt` objects. The function works exactly in the same way as `bio2bioDat` function.

4.3.7 Observation models: population

These type of models are useful when no assessment model is used in the next step of the MPM and management advice is just based on the population 'observed' in this step. `age2agePop`, `bio2bioPop` and `age2bioPop` are equal to their relatives in the previous section but in this case stock numbers, stock biomass and harvest are observed, with or without error, depending on the arguments given.

`NoObsStock`

This function is used when the advice is given independently to stock status. Therefore, we do not need to observe the population.

`perfectObs`

This function creates a `FLStock` from `FLBiol` and `FLFleetsExt` objects. The `FLBiol` and `FLFleetsExt` objects can be either aggregated in biomass or age structured and the returned `FLStock` object will have the same structure, but with unit and season dimensions collapsed. This function does not introduce any observation uncertainty in the observation of the different quantities stored in the `FLStock` or `FLFLeetsExt` objects. Slots relative to biological parameters are calculated averaging across units and seasons, those relative to catch are calculated summing up across units and seasons, and numbers at age or biomass are taken from the start of the first season, except recruitment that is obtained summing up the recruitment produced along seasons. Finally, fishing mortality is calculated numerically from numbers at age and natural mortality.

`age2agePop`

This function operates exactly in the same way as its counterpart in the previous section, `age2ageDat`, but it also fills `stock.n`, `stock.wt`, `stock` and `harvest` slots:

`stock.n`: First, the numbers at age are calculated as in `perfectObs` function and then 2 sources of uncertainty are introduced, as it is done in landings and discards at age. The error attributed to aging error is given by the same argument as in landings and discards at age, `ages.error`. The second uncertainty is introduced in the same way but by different argument, `stk.nage.error`.

`stock.wt`: First, the weight at age is calculated as in `perfectObs` function and then 2 sources of uncertainty are introduced, as it is done in weight at age of landings but replacing landings by stock numbers at age. The error attributed to aging error is given by the same arguments as in landings, `ages.error`. The second uncertainty is introduced in the same way but by different argument, `stk.wgt.error`.

`stock`: This is equal to the sum of the product of `stock.n` and `stock.wt`.

`harvest`: Harvest is numerically calculated from stock numbers at age and natural mortality.

`bio2bioPop`

This function operates exactly in the same way as its counterpart in the previous section `bio2bioDat` but it also fills `stock` and `harvest` slots:

`stock`: Stock biomass is calculated multiplying `n` and `wt` slots in the `FLBiol` object and summing up along seasons (note that unit dimension is always equal to 1 in populations aggregated in biomass). After, that uncertainty in the observation is introduced multiplying the obtained biomass by the argument `stk.bio.error`, which is an `FLQuant` with dimension `[quant = 1, year = ny, unit = 1, season = 1, area = 1, iter = ni]`

`harvest`: Harvest is calculated as the ratio between catch and stock biomass.

`age2bioPop`

This function operates exactly in the same way as its counterpart in the previous section `age2bioDat`, but it also fills `stock` and `harvest` slots. These two slots are calculated as in `bio2bioPop` function but summing up along ages in the case of `stock` slot.

4.3.8 Observation models: abundance indices

Currently, there are 2 functions that simulate abundance indices, one that generates age structured abundance indices `ageInd` and a second one that generates abundance indices in biomass `bioInd`. The last one can be applied to both age structured and biomass dynamics populations. In both cases a linear relationship between the index and the abundance is assumed being the catchability q the slope, i.e:

$$I = q \cdot N \quad \text{or} \quad I = q \cdot B$$

`ageInd`: age index observation model

Age structured abundance indices are obtained multiplying the slot `n` of `FLBiol` with the catchability of the index (`catch.q` in `FLIndex` object). The `FLIndex` is an input object and the `index` slot is yearly updated. Two sources of uncertainty are introduced, one related to aging error and a second one related to random variation. Aging error is the same as in the observation of landings at age and the argument is the same `ages.error`. Afterwards, the second source of uncertainty is introduced multiplying the index by the slot `index.var` of the `FLIndex` object. The indices do not need to cover the full age or year ranges.

`bioInd`: biomass index observation model

Biomass abundance indices are generated in the same way as age structured indices but without the error associated to age.

`NoObsIndex`: no index observation

This function is used when abundance indices are not required.

4.3.9 Observation models: fleets

At this point there are no functions to observe the fleets, their catch or catch at age is just observed in an aggregated way in the functions defined in previous section.

4.3.10 Management advice models

Different management advice models have been implemented. Some of them are methods generally applicable (e.g. `fixedAdvice`, `annualTAC`, `IcesHCR`, `annexIVHCR`, `CFPMSYHCR`, `F2CatchHCR`, `MAPHRC` and `MultiStockHRC`), whereas others are designed specifically for particular case studies (e.g. `FroeseHCR`, `ghlHCR`, `aneHCRE`, `neaMAC_ltmp`, `little2011HCR`, `pidHCR` and `pidHCRtarg`). All these rules are single-stock, apart from `MAPHRC` and `MultiStockHRC`, which are multi-stock harvest control rules.

`fixedAdvice`: fixed advice model

This function is used when the advice is fixed and independent to the stock status. TAC or TAE values should be given as input in the `advice` object.

`annualTAC`: annual TAC model

This function mimics the typical harvest control rule (HCR) used in recovery and management plans implemented in Europe. The function is a wrapper of the `fwd` function in `FLash` library. As `fwd` is only defined for age structured populations, within `FLBEIA` a new function `fwdBD` has been coded. `fwdBD` is a tracing of `fwd` but adapted to work with populations aggregated in biomass. The advice is produced in terms of catch (i.e TAC).

The call to `annualTAC` function within `FLBEIA` is done as:

```
annualTAC(stocks, advice, advice.ctrl, year, stknm, ...)
```

If the management is being running in year `y`, the function works as follows:

1. Project the observed stock one year forward from 1st of January of year y up to 1st of January of year `y+1` (intermediate year).
2. Apply the HCR and get the TAC for year `y+1`. Depending on the definition of the HCR the stock could be projected several years forward.

`advice.ctrl[[stock.name]]` for `annualTAC`

`HCR.model: 'annualTAC'`.

`nyears:` Number of years to project the observed stock from year `y-1`.

`wts.nyears:` Number of historic years to be used in the average of biological parameters. The average is used in the projection of biological parameters.

`fbar.nyears:` Number of historic years to be used in the average of selection pattern. The average is used in the projection of selection pattern.

`f.rescale:` Logical. If `TRUE` rescale to status quo fishing mortality.

`disc.nyears:` Number of years over which to calculate mean for `discards.n` and `landings.n` slots.

`fwd.ctrl:` Element of class `fwdControl`. For details on this look at the help page in `FLash` object. The only difference is the way the years are introduced. As this object is defined before simulation and it is applied year by year, the definition of the year should be dynamic. Thus the following convention has been taken:

- `year = 0` indicates the year when management is taking place, (intermediate year).
- `year = -1` corresponds with one year before the year when management is taking place. In this case, whithin `annualTAC` function, coincides with the year up to which data is available, (data year). Then, -2 would indicate 2 years before,-3 would indicate 3 years before and so on.
- `year = 1` corresponds with one year after the year when management is taking place. In this case, whithin `annualTAC` function, coincides with the year for which management advice is going to be produced, (TAC year). Then, 2 would indicate 2 years after the year when management is taken place, 3 would indicate 3 years after and so on.

In this way, within the simulation, each year, the intermediate year is summed up to the `year` in the original control argument and the correct year names are obtained.

`AdvCatch:` Vector with a logic value for each year. TAC is given in terms of catch, if `TRUE`, or landings, if `FALSE`.

`sr:` The stock recruitment relationship used to project the observed stock forward, not needed in the case of population aggregated in biomass. `sr` is a list with 3 elements, `model, params` and `years`. `model` is mandatory and the other 2 are complementary, if `params` is given `years` is not necessary. `model` can be any stock-recruitment model defined for `FLSR` class. `params` is a `FLPar` model an

if specified it is used to parameterized the stock-recruitment model. `years` is a numeric named vector with 2 elements `'y.rm'` and `'num.years'`, for example `c(y.rm = 2, num.years = 10)`. This element is used to determine the observeds years to be used to estimate the parameters of the stock recruitment relationship. In the example the last 2 observations will be removed and starting from the year before to the last 2 observed years 10 years will be used to estimate the stock-recruitment parameters.

`growth.years`: This argument is used only for stocks aggregated in biomass and it indicates the years to be used in the estimation of annual population growth. This growth is used to project the population forward. `growth.years` is a numeric named vector with 2 elements `'y.rm'` and `'num.years'` which play the same role played in `sr[['years']]` argument defined in the previous point.

`IcesHCR`: ICES harvest control rule

The function represents the HCR used by ICES to generate TAC advice in the MSY framework. It is a biomass based HCR, where the TAC advice depends on F in relation to several reference points: a biomass that triggers the F reduction ($\mathtt{B}_{trigger}$), the limit biomass below which there is a high risk of impaired recruitment ($\mathtt{B}_{lim}$) and fishing mortality that leads to MSY ($\mathtt{F}_{MSY}$).

Current function calls `annualTAC`, given an F objective calculated as:

$$F_{target} = \begin{cases} 0 & \text{, if } B < B_{lim} \\ F_{MSY} \cdot B/B_{trigger} & \text{, if } B < B_{trigger} \\ F_{MSY} & \text{, if } B \geq B_{trigger} \end{cases} \tag{13}$$

The call to `IcesHCR` function within `FLBEIA` is done as:

```
IcesHCR(stocks, advice, advice.ctrl, year, stknm, ...)
```

`advice.ctrl[[stock.name]]` for `IcesHCR`

`HCR.model: 'IcesHCR'`.

`nyears`: Number of years to project the observed stock from year `y-1`.

`wts.nyears`: Number of historic years to be used in the average of biological parameters, if missing last 3 years are used. The average is used in the projection of biological parameters.

`fbar.nyears`: Number of historic years to be used in the average of selection pattern, if missing last 3 years are used. The average is used in the projection of selection pattern.

`f.rescale`: Logical. If `TRUE` rescale to status quo fishing mortality.

`ref.pts`: Matrix of dimension [3,it], where rows contain values for B_{lim}, $B_{trigger}$ and F_{MSY}, and `colnames(ref.pts) = c(Blim, Btrigger, Fmsy)`.

`AdvCatch`: Vector with a logic value for each year. TAC is given in terms of catch, if `TRUE`, or landings, if `FALSE`.

`intermediate.year`: Sets how to calculate the catches in the intermediate year. If it is set to `'Fsq'`, then the catches are estimated based on the last estimated F; whereas if other value set, the catches are set to the TAC adviced for this intermediate year. This second approach is used for the cases when the assessment is carried out including also the information on this intermediate year, as is the case for the Bay of Biscay anchovy.

`sr`: The stock recruitment relationship used to project the observed stock forward, not needed in the case of population aggregated in biomass. `sr` is a list with 3 elements, `model, params` and `years`. For more details see parameter description in `annualTAC` (above).

`growth.years`: This argument is used only for stocks aggregated in biomass and it indicates the years to be used in the estimation of annual population growth. This growth is used to project the population forward. For more details see parameter description in `annualTAC` (above).

`FroeseHCR`: Froese harvest control rule

This function recreates the HCR defined in the paper by **?**, which is a biomass based HCR. TAC advice is calculated depending on perceived biomass in relation to biological reference points as follows:

$$TAC = \begin{cases} 0 & \text{, if } B < B_{trigger} \\ MSY \cdot \beta \cdot 1/(1-\alpha_0) * (-\alpha_0 + B/B_{target}) & \text{, if } B_{trigger} \leq B < B_{target} \\ MSY \cdot \beta & \text{, if } B \leq B_{target} \end{cases} \tag{14}$$

where $B_{trigger} = \alpha_0 \cdot B_{MSY}$ and $B_{target} = \alpha_1 \cdot B_{MSY}$.

The call to `annualTAC` function within `FLBEIA` is done as:

`FroeseHCR(stocks, advice, advice.ctrl, year, stknm,...)`

`advice.ctrl[[stock.name]]` **for** `FroeseHCR`

`HCR.model: 'FroeseHCR'`.

`ref.pts:` Matrix of dimension [5,it], where rows contain values for B_{MSY}, MSY, α_0, α_1 and β, and `colnames(ref.pts) = c(Bmsy, MSY, alpha_0, alpha_1, beta)`.

`annexIVHCR`: ICES Annex IV harvest control rule

This funcion emulates the HCR used by the European Comission and ICES to generate the TAC advice for data poor stocks. TAC advice is calculated depending on previous year TAC and the trend of an available index as follows:

$$TAC_{y+1} = \gamma \cdot TAC_y \tag{15}$$

$$\gamma = \begin{cases} 1-\beta & \text{, if } B_{now}/B_{ref} \leq 1-\alpha \\ 1 & \text{, if } 1-\alpha < B_{now}/B_{ref} < 1+\alpha \ \& \ type = 2 \\ \beta/\alpha \cdot (B_{now}/B_{ref} - 1) + 1 & \text{, if } 1-\alpha < B_{now}/B_{ref} < 1+\alpha \ \& \ type = 4 \\ 1+\beta & \text{, if } B_{now}/B_{ref} \geq 1+\alpha \end{cases} \tag{16}$$

where: $B_{now} = (I_{y-1} + I_{y-2})/2$ and $B_{ref} = (I_{y-3} + I_{y-4} + I_{y-5})/3$.

The call to `annexIVHCR` function within `FLBEIA` is done as:

`annexIVHCR(indices, advice, advice.ctrl, year, stknm,...)`

`advice.ctrl[[stock.name]]` **for** `annexIVHCR`

`HCR.model: 'annexIVHCR'`.

`index:` Either the name or the position of the index in FLIndices object.

`ref.pts:` Matrix of dimension [2,it], where rows contain values for α and β, and `colnames(ref.pts) = c(alpha, beta)`.

`type:` Numeric (options 2 or 4). This parameter determinines the value of γ in Equation **??**.

`ghlHCR`: Greenland halibut harvest control rule

This function mimics thee model-free HCR used in the management of greenland-halibut in NAFO. TAC advice is calculated depending on previous year TAC and the trends of three indices available for the stock as follows:

$$TAC_{y+1} = TAC_y + \lambda \cdot slope \tag{17}$$

$$\lambda = \begin{cases} \alpha_0 & \text{, if } slope < 0 \\ \alpha_1 & \text{, if } slope > 0 \end{cases} \tag{18}$$

where λ value has the following additional constraint: $1-\beta \leq \lambda \leq 1+\beta$, and $slope$ is the mean of the slopes obtained when calculating a linear model for each of the indices.

The call to `ghlHCR` function within `FLBEIA` is done as:

`ghlHCRC(indices, advice, advice.ctrl, year, stknm,...)`

`advice.ctrl[[stock.name]]` **for** `ghlHCR`

`HCR.model: 'ghlHCR'`.

`ref.pts:` Matrix of dimension [3,it], where rows contain values for α_0, α_1 and β, and `colnames(ref.pts) = c(alpha_0, alpha_1, beta)`.

aneHCRE: Bay of Biscay anchovy first long term management plan - HCRE

The function recreates the HCR used in the Bay of Biscay anchovy first long term management plan, where HCR was known as Rule E.

TAC advice is calculated depending on perceived biomass in relation to biological reference points as follows:

$$TAC = \begin{cases} 0 & \text{, if } SSB \leq 24,000\text{tons} \\ 7,000 & \text{, if } 24,000 < SSB < 33,000\text{tons} \\ hr \cdot SSB & \text{, if } SSB \geq 33,000\text{tons} \end{cases} \tag{19}$$

whith the following additional constraint: $TAC \leq 33,000$ tons.

The call to `aneHCRE` function within `FLBEIA` is done as:

```
aneHCRE(stocks, advice, advice.ctrl, year, stknm,...)
```

neaMAC_ltmp: Northeast Atlantic mackerel long term management plan

This function emulates the HCR used in the north-east atlantic mackerel long term management plan. It is a particular case of the IcesHCR.

F2CatchHCR: F to catch harvest control rule

This function transforms the fishing mortality advice given as input data to catch advice without any other restriction. The function is a copy-paste from `IcesHCR`, but in this case target F is directly taken from `ref.pts['Ftarget',year+1,]`.

little2011HCR: Little's harvest control rules

This function mimics the HCR defined in the paper by **?**, with an additional constraint, C_{max}, not to allow very high catches. This constraint can be turned off setting C_{max} to a very high value or `Inf`.

TAC advice is calculated depending on an index in relation to biologicalsome reference points as follows:

$$TAC = min(C_{targ} \cdot max(0, (I_y - I_{lim})/(I_{targ} - I_{lim})), C_{max}) \tag{20}$$

where: C_{targ} and C_{max} correspond to target and maximum catches, respectively, I_y corresponds to the mean value of the index in the last two years and I_{targ}, I_{lim}) are reference values with respect to the index.

The call to `little2011HCR` function within `FLBEIA` is done as:

```
little2011HCR(indices, advice, advice.ctrl, year, stknm, ...)
```

advice.ctrl[[stock.name]] for little2011HCR

`HCR.model: 'little2011HCR'`.

`index:` Either the name or the position of the index in FLIndices object.

`ref.pts:` Matrix of dimension [4,it], where rows contain values for C_{targ}, I_{lim}, I_{targ} and C_{max}. and `colnames(ref.pts) = c(Ctarg, Ilim, Itarg, Cmax)`.

pidHCR and pidHCRtarg: Pomaerede's harvest control rules

These functions recreates the model free HCRs used for hake and wich are defined in the paper by **?**.

The call to these functions within `FLBEIA` is done as:

```
pidHCR(indices, advice, advice.ctrl, year, stknm, ...)
pidHCRtarg(indices, advice, advice.ctrl, year, stknm, ...)
```

advice.ctrl[[stock.name]] for pidHCR and pidHCRtarg

`HCR.model: 'pidHCR'` or `'pidHCRtarg'`.

`index:` Either the name or the position of the index in FLIndices object.

`ref.pts:` For `pidHCR`, matrix of dimension [5,it], where rows contain values for K_p, K_i, K_d, τ and α; whereas for function `pidHCRtarg` it has dimension [6,it] with an additional row for I_{targ} values. `colnames(ref.pts) = c(Kp, Ki, Kd, tau, alpha, Itarg)`.

MAPHRC: harvest control rule for multi-annual management plans

This function emulates the HCR proposed by the European Comission for the evaluation of multi-annual management plans (MAPs) in 2015. This HCR is specially designed to fulfill the requirements of MAPs for North Western Waters and only works for age-structured stocks.

The call to `MAPHRC` function within `FLBEIA` is done as:

```
MAPHRC(stocks, advice, advice.ctrl, year, stknm, ...)
```

`advice.ctrl[[stock.name]]` for `MAPHRC`

`HCR.model: 'MAPHRC'`.

`wts.nyears:` Number of historic years to be used in the average of biological parameters, if missing last 3 years are used. The average is used in the projection of biological parameters.

`fbar.nyears:` Number of historic years to be used in the average of selection pattern, if missing last 3 years are used. The average is used in the projection of selection pattern.

`f.rescale:` Logical. If `TRUE` rescale to status quo fishing mortality.

`ref.pts:` FLQuant of dimension `[quant = 4, year = ny, unit = 1, season = 1, area = 1, iter = ni]`, where `quant` dimension contains values for B_{pa}, F_{target}, C_{up} and C_{lo}, and `dimnames(ref.pts)[1] = c(Bpa, Ftarget, Cup, Clo)`.

`N:` Numeric value, corresponding to the number of years to recover SSB.

`AdvCatch:` Vector with a logic value for each year. TAC is given in terms of catch, if `TRUE`, or landings, if `FALSE`.

`sr:` The stock recruitment relationship used to project the observed stock forward, not needed in the case of population aggregated in biomass. `sr` is a list with 3 elements, `model, params` and `years`. For more details see parameter description in `annualTAC` (above).

CFPMSYHCR: flexible harvest control rule for multi-annual management plans

This function is a version of the `MAPHCR`, adapting it to allow flexibility in the year Fmsy is achieved. The user can specify the year in which you aim to reach F_{MSY}, with a linear transition between F_{sq} to F_{MSY} in the intervening years.

The call to `CFPMSYHCR` function within `FLBEIA` is done as:

```
CFPMSYHCR(stocks, advice, advice.ctrl, year, stknm, ...)
```

`advice.ctrl[[stock.name]]` for `CFPMSYHCR`

`HCR.model: 'CFPMSYHCR'`.

`wts.nyears:` Number of historic years to be used in the average of biological parameters, if missing last 3 years are used. The average is used in the projection of biological parameters.

`fbar.nyears:` Number of historic years to be used in the average of selection pattern, if missing last 3 years are used. The average is used in the projection of selection pattern.

`f.rescale:` Logical. If `TRUE` rescale to status quo fishing mortality.

`ref.pts:` FLQuant of dimension `[quant = 5, year = ny, unit = 1, season = 1, area = 1, iter = ni]`, where `quant` dimension contains values for B_{pa}, F_{target}, Yr_{tg}, C_{up} and C_{lo}, and `dimnames(ref.pts)[1] = c(Bpa, Ftarget, Yrtg, Cup, Clo)`.

`N:` Numeric value, corresponding to the number of years to recover SSB.

`AdvCatch:` Vector with a logic value for each year. TAC is given in terms of catch, if `TRUE`, or landings, if `FALSE`.

`sr:` The stock recruitment relationship used to project the observed stock forward, not needed in the case of population aggregated in biomass. `sr` is a list with 3 elements, `model, params` and `years`. For more details see parameter description in `annualTAC` (above).

`MultiStockHRC`: multi-stock harvest control rule

This function produces TAC advice for several stocks simultaneously, this HCR is based on `IcesHCR`. It uses a fishing mortality target and an upper bound to conciliate the TAC advices. In the case of stocks without exploitation rate estimates, then it uses the catch. At present this function only works with single iterations.

TAC advice is calculated depending on fishing mortality in relation to biological reference points of all the stocks. First, for each stock we calculate the single stock Ftarget depending on its status in relation to the BRPs.

$$F_{target} = \begin{cases} 0 & \text{, if } B < B_{lim} \\ F_{MSY} \cdot B/B_{trigger} & \text{, if } B_{lim} \leq B < B_{trigger} \\ F_{MSY} & \text{, if } B \geq B_{trigger} \end{cases} \quad (21)$$

Second, we calculate the ratio between F_{target} and F_{sq} and calculate the maximum:

$$F_{adv0}[stock.name] = \lambda_0 \cdot F_{sq} | \lambda_0 = max_{i \in names(biols)}(F_{target}/F_{sq})[i] \quad (22)$$

Therefore, there is only one stock for which $F_{adv0} = F_{target}$ and for the rest $F_{adv0} > F_{target}$. Third, we calculate the ratio between Fupp and Fsq and calculate the minimum:

$$x_{st} = F_{upp}[stock.name]/F_{adv0}[stock.name]\lambda_0 \cdot F_{sq} \forall st \in names(biols) \quad (23)$$

$$\begin{cases} \text{If } x_{st} \geq 1 \forall st, & \lambda_1 = 1 \\ \text{If } st \& x_{st} < 1, & \lambda_1 = min(F_{upp}[st]/F_{adv0}[st] \\ & \&F_{adv1}[st] = \lambda_1 \cdot F_{adv0}[st] \end{cases} \quad (24)$$

Therefore, there is only one stock for which $F_{adv0} = F_{target}$ and for the rest $F_{adv0} > F_{target}$. Finally,

$$F_{adv}[stock.name] = \lambda_1 \cdot \lambda_0 \cdot F_{sq} \quad (25)$$

And the TAC for each stock is calculated based on advised fishing mortality (i.e. $F_{adv}[stock.name]$).

The call to `annualTAC` function within `FLBEIA` is done as:

```
MultiStockHCR(stocks, indices, advice, advice.ctrl, year, stknm,...)
```

In relation to [IcesHCR this new HCR has two additional arguments:

`advice.ctrl[['stocksInHCR']]:` A vector with the name of the stocks that are taken into account in the calculation of advice.

`advice.ctrl[[stock.name]][['ref.pts']]:` A new row in the matrix with `Fupp` value.

4.4 Fourth level functions

These functions are called by the third level functions and, for the time being, are the functions in the lowest level within `FLBEIA`.

4.4.1 Stock-recruitment relationships

Stock-recruitment relationships are used, for example, within `ASPG` and `annualTAC` functions. The stock-recruitment relationship used in `ASPG` is defined in the slot `model` of `FLSRsim` object and it defines the true recruitment dynamics of the stocks. Within `annualTAC`, the stock-recruitment relationship used is defined in:

```
advice.ctrl[[stock.name]][['sr']][['model']]
```

element and it describes the 'observed' stock-recruitment dynamics (used) in the management process.

In `FLCore` package there are several stock-recruitment relationships already defined and all can be used within `FLBEIA`. Some of the functions available are:

`geomean:` Recruitment is independent of the stock and equal to the geometric mean of historical period.

$$R = \alpha = \sqrt[n]{R_1 \cdot \ldots \cdot R_n}$$

`bevholt:` Beverton and Holt model with the following parameterization:

$$R = \frac{\alpha \cdot SSB}{(\beta + SSB)}$$

where α is the maximum recruitment (asymptotically) and β is the stock level needed to produce the half of maximum recruitment $\alpha/2$ $(\alpha, \beta > 0)$.

`ricker`: Ricker stock-recruitment model fit with the following parameterization:

$$R = \alpha \cdot SSB \cdot e^{-\beta \cdot SSB}$$

where α is related to productivity and β to density dependence. α is the recruit per stock unit at small stock levels. $(\alpha, \beta > 0)$.

`segreg`: Segmented regression stock-recruitment model fit:

$$R = \begin{cases} \alpha \cdot SSB & \text{, if } SSB < \beta \\ \alpha \cdot \beta & \text{, if } SSB \geq \beta \end{cases}$$

where α is the slope of the recruitment for stock levels below β and $\alpha \cdot \beta$ is the mean recruitment for stock levels above β $(\alpha, \beta > 0)$.

`shepherd`: Shepherd stock-recruitment model fit:

$$R = \alpha \cdot \frac{SSB}{(1 + (S/\beta)^{\gamma})}$$

This model generalizes Beverton and Holt and Ricker models ($\gamma = 1$ corresponds with Beverton and Holt model, $\gamma > 1$ takes a Ricker-like shape and with $\gamma < 1$ the curve rises indefinitely).

`bevholtAR1, rickerAR1, segregAR1`: Beverton and Holt, Ricker and Segmented regression stock-recruitment models with autoregressive normal log residuals of first order. In the model fit the corresponding stock-recruitment model is combined with an autoregressive normal log likelihood of first order for the residuals. If R_t is the observed recruitment and $\hat{R}_t$ is the predicted recruitment, an autoregressive model of first order is fitted to the log-residuals, $x_t = log(R_t/\hat{R}_t)$.

$$x_t = \rho \cdot x_{t-1} + \varepsilon$$

where $\varepsilon \sim N(0, \sigma^2_{ar})$.

`cushing`: Cushing stock recruitment model fit:

$$R = \alpha \cdot SSB^{\beta}$$

where $\alpha, \beta > 0$.

`bevholtSV, rickerSV, segregSV, shepherdSV, cushingSV`: Beverton and Holt, Ricker, Segemented regression, Shepherd and Cushing stock-recruitment models with α and β parameterisation converted into steepness and virgin biomass (s and v).

`rickerCa`: Ricker stock-recruitment model with covariates, parameterised as:

$$R = \alpha \cdot (1 - \gamma \cdot covar) \cdot SSB \cdot e^{-\beta \cdot SSB}$$

Additionally, the following stock-recruitment relationships are defined in `FLBEIA` package:

`hockstick`: Hockey stick stock-recruitment model fit:

$$R = \begin{cases} \alpha \cdot S & \text{, if } SSB < \beta \\ \alpha \cdot \beta & \text{, if } SSB \geq \beta \end{cases}$$

where α is the slope of the recruitment for stock levels below β and $\alpha \cdot \beta$ is the mean recruitment for stock levels above β $(\alpha, \beta > 0)$.

`redfishRecModel`: Redfish recruitment model developed by Benjamin Planque, with the formula:

$$R_y = redfishRec(R_{y-1}, \sigma, minrec, maxrec) \cdot \begin{cases} \frac{SSB}{\alpha} & \text{, if } SSB < \alpha \\ 1 & \text{, if } SSB \geq \alpha \end{cases}$$

Being:

$$redfishRec(R_{y-1}, \sigma, minrec, maxrec) = R_{y-1} + rnorm(n = 1, mean = 0, sd = \sigma)$$

$$minrec \leq redfishRec(R_{y-1}, \sigma, minrec, maxrec) \geq maxrec$$

Where R_{y-1} corresponds to previous year's recruitment, γ the standard deviation of the historic recruitment and `minR`, `maxR` to the minimun and maximum recruitments historically observed, respectively.

`ctRec`: constant recruiment. There is not recruitment modelling and therefore expected recruitment values for the projection period has to be fixed a priori.

There could be more stock-recruitment relationships defined in `FLCore` or `FLBEIA`, thus, if you are interested in using a model not defined here take a look at `SRModels` help page in `FLCore` package. New stock-recruitment models to be used in `FLSRsim` class can be defined in two ways:

1. Using a formula in slot `model`:
$$rec \sim \Phi(X)$$
where Φ is a function of `ssb` and parameters and covariates stored in `params` and `covar` slots respectively.
2. Defining a function in `R`, `foo <- function(X)`, and using the name of the function, `foo`, in slot `model`. The function arguments must be among `ssb` and parameters and covariates stored in `params` and `covar` slots respectively.

4.4.2 Catch production functions

The catch production functions can be different for the same third level effort model. Currently, there are three catch production functions available. The first two correspond with Cobb-Douglas production functions (**??**) but in one case the model operates at stock level and in the second one at age class level. The last one is used when information on effort is lacking and consequently catches are set independently from effort.

`CobbDouglasBio`: Cobb-Douglas production function at stock level

The total catch of the fleet is calculated according to the Cobb-Douglas production function:

$$C = q \cdot E^{\alpha} \cdot B^{\beta} \tag{26}$$

where C denotes total catch and B total biomass (both in weight), q the catchability and E the effort. α and β are the elasticity parameters associated to labor and capital (biomass in this case), respectively. These parameters are associated to the existing technology.

As α and β parameters depend on the stock and the technology, Cobb-Douglas function is applied at metier level. Thus, the catch of a certain fleet f is given by:

$$C_f = \sum_{m \in M_f} q_{f,m} \cdot B^{\beta_{f,m}} \cdot (E_f \cdot \delta_{f,m})^{\alpha_{f,m}} \tag{27}$$

where M_f represents the set of metiers of fleet f and δ the effort share among metiers.

Derivation of Catch-at-age. Once the total catch is calculated, it is divided into catch at age using selectivity at age, $s_{a,f,m}$, and biomass at age in the population, B_a:

$$C_{a,f,m} = \frac{C_{f,m}}{\sum_a s_{a,f,m} \cdot B_a} \cdot s_{a,f,m} \cdot B_a \tag{28}$$

Derivation of Equation **??**:

- If the whole population were accessible to the gear, the catch of age a would be:
$$s_{a,f,m} \cdot B_a$$
- Thus, if the whole population were accessible to the gear, the total catch we could obtain would be:
$$\sum_a s_{a,f,m} \cdot B_a$$
- But, the actual total catch is C_f, so theoretically the proportion of the population that have been accessible is[1]:
$$\frac{C_{f,m}}{\sum_a s_{a,f,m} \cdot B_a}$$
- Then, if we assume the population is homogeneously distributed we arrive to Equation **??**.

[1] If all the age classes were not accessible or completely accessible we would replace $s_{a,f,m}$ by $s_{a,\acute{f},m} = \gamma_{a,f,m} \cdot s_{a,f,m}$ where $\gamma_{a,f,m}$ is the proportion of individuals of age a accessible to metier m in fleet f.

The catch at age is then further disaggregated in landings- and discards-at-age using landings' and discards' specific selectivity:

$$L_{a,f,m} = \frac{sl_{a,f,m}}{s_{a,f,m}} \cdot C_{a,f,m} \quad \text{and} \quad D_{a,f,m} = \frac{sd_{a,f,m}}{s_{a,f,m}} \cdot C_{a,f,m} \tag{29}$$

`CobbDouglasAge`: Cobb-Douglas production function at age-class level

The catch of the fleets is calculated according to the Cobb-Douglas production function applied at age-class level, i.e.:

$$C = \sum_a C_a = q_a \cdot E^{\alpha_a} \cdot B_a^{\beta_a} \tag{30}$$

where C denotes catch and B biomass (both in weight), q the catchability, E the effort and a the subscript for age. α and β are the elasticity parameters associated to labor and capital (biomass in this case), respectively. These parameters are associated to the existing technology.

As α and β parameters dependent on age classes and technology, Cobb-Douglas function is applied at metier level. Thus, the catch of a certain fleet f is given by:

$$C_f = \sum_a C_{a,f} = \sum_{m \in M_f} \sum_a q_{a,f,m} \cdot B_a^{\beta_{a,f,m}} \cdot (E_f \cdot \delta_{f,m})^{\alpha_{a,f,m}} \tag{31}$$

where M_f represents the set of metiers of fleet f, δ the effort share among metiers and m is the subscript that indicates the metier.

`seasonShare`: Catches estimation given season share allocation by metier

In case that there is no information on the effort, and therefore effort is not limiting the catches, the catch at age of each metier can be calculated according to a previously defined season share allocation by metier. This function is only valid for metiers which target only one stock. Alternatively, the seasonal share for one stock can be set equal to the one of a reference fleet (usually one which has information on effort).

Two arguments need to be declared as elements of `fleets.ctrl` if this function is used:

`effort.model`: 'fixedEffort'. This argument must be declared at fleet level (i.e. `fleets.ctrl[[fleet.name]]$effort.model`)

`catch.model`: Argument is used to specify the catch production function that will be used to generate the catch and it must be declared at fleet and stock level (i.e. `fleets.ctrl[[fleet.name]][[stock.name]]$catch.model`). That catch production model corresponds with a fourth level function (see Section **??**).

`catch.dependence`: This argument needs to be set only when aiming to set the seasonal share of one stock equal to the same stock in other reference fleet and it must be declared at fleet and stock level (i.e. `fleets.ctrl[[fleet.name]][[stock.name]]$catch.dependence`). Value has to be set equal to the name of this reference fleet.

4.4.3 Costs functions

Cost functions have been developed in order to be used within `fleets.om`. As cost structure could differ among fleets it has been defined as fourth level function and it works at fleet level. In principle, it could be useful in both tactic and strategic dynamics of fleets.

`TotalCostsPower`: Total costs power function

This function sums up the fixed costs (FxC) and the power functions of cost per unit of effort ($CostPUE$), crew share per unit of landings ($CSPUL$) and capital cost per unit capital ($CapCostPUC$), mathematically:

$$\begin{aligned} Cost_f = & FxC_f + \sum_m (CostPUE_{f,m} \cdot E_f \cdot \tau_{f,m})^{\gamma_{1f,m}} + \\ & \sum_{st} \sum_m CSPUL_{f,m,st} \cdot L_{f,m,st}^{\gamma_{2f,m,st}} + CapCostPUC_f \cdot Cap_f^{\gamma_{3f}} \end{aligned} \tag{32}$$

The fixed cost are given at fleet level, f, cost per unit of effort at metier level, m, and crew share at fleet, metier and stock, st, level. $\gamma_{1f,m}$ is the exponent of effort at fleet and metier level in cost of effort addend, $\gamma_{2f,m,st}$ the exponent of landing at fleet, metier and stock level in crew share cost addend and γ_{3f} is the exponent of capital at fleet level in capital cost addend.

5 Smart conditioning

5.1 Functions

FLBEIA requires a number of input arguments to run a simulation; such as biols, SRs, BDs, fleets, covars, indices, advice, main.ctrl, biols.ctrl, fleets.ctrl, covars.ctrl, obs.ctrl, assess.ctrl and advice.ctrl. These objects contain biological and economical historical and projection data, and also point the functions that are going to be used by the model. Here we introduce and explain the functions that have been generated to facilitate the creation of these objects:

create.biols.data: It generates an FLBiol object for each stock, and includes all of them in a FLBiols object. It returns an object that could be used as biols argument in FLBEIA function. The function requires historical data of weight, abundance, natural mortality, fecundity and spawning. In the projection years, natural mortality, fecundity and spawning, are assumed equal to the average of the historical years that the user specifies in stk_biol.proj.avg.yrs. The descriptions and format of the arguments required by the function are presented in Table **??**.

create.SRs.data: It generates a list with FLSRsim objects and returns an object that could be used as SRs argument in FLBEIA function. This function does not calculate stock-recruitment function's parameter values; therefore, they must be calculated previously by the user. In case that the proportion of spawning per season is not defined in the projection years, then it is assumed equal to the average of the historical years range that the user defines. If uncertainty is not an input, then there is not uncertainty. The descriptions and format of the arguments required by the function are presented in Table **??**.

create.BDs.data: It generates a list with FLBDsim objects. It returns an object that could be used as BDs argument in FLBEIA function. This function does not calculate biomass dynamics function's parameter values, so they must be introduced by the user. If uncertainty is not an input, then the function assumes no uncertainty. The descriptions and format of the arguments required by the function are presented in Table **??**.

create.fleets.data: It generates an FLFleet object for each fleet and includes all of them in an FLFleets object. It returns an object that could be used as fleets argument in FLBEIA function. The function requires historical data of effort per fleet, effort share between métiers and, landings and weight at age per stock. Notice that the input data of landingsẃeight has the same name as stock weight input in create.biols.data. In case that discards data are available, then its weight is assumed the same as for landings. The descriptions and format of the arguments required by the function are presented in Table **??**. The function assumes that when historical data of effort, fixed cost, capacity or crewshare are introduced, then the projection values of each of them are the average of the years that the user sets in fl.proj.avg.yrs; in the case of effort share and variable cost per metier, is set in fl.met_proj.avg.yrs; and in the case of landings at age, discards at age and price, in fl.met.stk_proj.avg.yrs. If the Cobb-Douglas parameters, alpha, beta and q (see more information on Section **??**), are not introduced as inputs, then they are created by the calculate.CBparam function, where it assumes that alpha and beta are equal to 1 and q is the ratio between total catch and effort per metier multiplied by the stock abundance.

create.indices.data: It generates a list with all the stocks and for each stock a list with FLIndex objects. It returns an object that could be used as indices argument in FLBEIA function. The descriptions and format of the arguments required by the function are presented in Table **??**.

create.advice.data: It generates a list with the advice for each of the stocks. It returns an object that could be used as advice argument in FLBEIA function. In case that the values of TAC and TAE are not introduced in the projection years, then the model assumes that they are equal to the average of the historical data that the user defines in stk_advice.avg.yrs. When quota share is not defined in the projection years, then the function calculates it for each stock as the ratio between catch per fleet and total catch. The descriptions and format of the arguments required by the function are presented in Table **??**.

create.biols.ctrl: It creates an object with the name of the growth function for each stock. The object that returns this function can be used as biols.ctrl argument in FLBEIA function.

create.fleets.ctrl: It creates an object with the fleet dynamics function that is applied for each fleet. The object that returns this function can be used as fleets.ctrl argument in FLBEIA function.

create.covars.ctrl: It creates an object with the function that is applied to the covariate. The object that returns this function can be used as covars.ctrl argument in FLBEIA function.

`create.obs.ctrl`: It creates a function with the observed function for each stock. The object that returns this function can be used as `obs.ctrl` argument in `FLBEIA` function.

`create.advice.ctrl`: I creates an object with the harvest control rule for each stock, and its parameter values. The object that returns this function can be used as `advice.ctrl` argument in `FLBEIA` function.

`create.assess.ctrl`: It creates an object with the name of the kind of assessment that is applied to each stock. The object that returns this function can be used as `assess.ctrl` argument in `FLBEIA` function.

We generate other functions to simplify the previous ones:

`create.list.stks.flqa`: It creates a list of `FLQuant` objects for each stock with the corresponding dimensions and dimension names. One of the dimensions in the `FLQuant` is age; in a range between the minimum and maximum age. The descriptions and format of the arguments required by the function are presented in Table **??**.

`create.list.stks.flq`: It creates a list of `FLQuant` objects for each stock with the corresponding dimensions and dimension names. One of the `FLQuant` is age, defined as `all`. The descriptions and format of the arguments required by the function are presented in Table **??**.

`calculate.CBparam`: It creates a list with the three Cobb-Douglas parameters: `alpha`, `beta` and `q`. It assumes that `alpha` and `beta` are equal to 1, and `q` is calculated as the ratio between total catch and the multiplication of effort per metier and stock abundance. The descriptions and format of the arguments required by the function presented in Table **??**.

5.2 Examples

Each test case has three folders: `data`, `R` and `plots`. In the folder called `data` are the input data in `csv` format. In the folder called `R` is the conditioning script, which: (i) creates the input objects for `FLBEIA` and saves them in the same folder with `RData` format; and (ii) runs the model and makes plots that are saved in the folder `plots`. In `R` folder, there is another folder called `results` with the output of the `FLBEIA` model in `RData` format.

1 stock, 1 fleet and 1 season: This test case analyzes the dynamics of a fictitious stock (`SBR`) and a fictitious fleet (`DLL`). The fleet has one metier, with the same name as the fleet (`DLL`). This case study is an example on how to include in the model: an assessment (XSA), observations data (disaggregated in ages) or elastic price.

The biological historical data available in this study range from 1990 to 2009 and `FLBEIA` is run from 1990 to 2025, with first year of projection in 2010. Biological data are described by stock abundance, weight, spawning, fecundity and natural mortality with three iterations, but only abundance and weight have some variability. Biological data are age specific in a range of ages between 1 and 12 years and with only one season. The minimum year to calculate the average `f` is set as 1 and the maximum as 12. The values of weight, spawning, fecundity and natural mortality in the projection period are assumed to be equal to the average between 2007 and 2009.

The fleet has historical information on effort, crewshare, capacity and fixed costs. There is only one metier, so effort share is one. Landings and discards, as well as Cobb-Douglas parameters (`alpha`,`beta`,`q`) values are introduced as input.

The model applies an age-structured population growth model and catch model. Beverton-holt autoregressive model is assumed as stock-recruitment relationship and uncertainty is not introduced in the projection.

2 stocks, 2 fleets and 4 seasons: This test case analyzes the dynamics of two fictitious stocks (`stk1` and `stk2`), with two fictitious fleets (`fl1` and `fl2`). Each of the fleets has two metiers (`met1` and `met2`) and all the metiers catch both stocks. This case study simulates the management using an ICES harvest control rule for stock `stk1` and annual TAC for `stk2`. The effort function of both fleets is different; fixed effort is assumed for fleet `fl1` and simple mixed fisheries behavior, limited by the minimum catch of both stocks, for fleet `fl2`. There is no assessment in this study and perfect observation is assumed.

The biological historical data available for both stocks range from 1990 to 2008. The simulation time covers from 1990 to 2025, with 2009 as the first projection year. Biological data are described by stock abundance, weight, spawning, fecundity and natural mortality. Biological data of stock `stk1` are age structured in a range from 0 to 15 years, but not for stock `stk2`, which is modelled in

biomass. The minimum age to calculate the average `f` for stock `stk1` is set as 1 and the maximum as 5. Stock `stk1` has 4 spawning seasons, while stock `stk2` only one, then we set different units for both. The values of weight, spawning, fecundity and natural mortality in the projection period are assumed to be equal to the average between 2006 and 2008 for both stocks.

Both fleets have historical information on effort and capacity, but only fleet `fl2` has fixed cost data. For each fleet and metier, effort share data are available and only for fleet `fl2` are variable costs. For each fleet and metier, there is age structured data for stock `stk1` and in biomass for `stk2`. Both fleets have landings and discards data for each stock, but in the case of stock `stk2` discards data of metier `met2` in fleet `fl1` are missing. Cobb-Douglas parameters (`alpha`,`beta`,`q`) are not introduced as input; therefore the function `create.fleets.data` calls `calculate.CBparam` function to calculate them.

Since the biological and economical historic data of stock `stk1` are age specific, then the model allows using age-structured population growth and catch model, while in the case of stock `stk2` they must be based on biomass. Beverton-Holt model is applied as stock-recruitment relationship for stock `stk1`, with 4 spawning seasons, and Pella-Tomlinson biomass dynamics model for stock `stk2`. In both cases the parameters in the projection period are introduced as input.

6 Output summary

There are several functions available to summarise the results of the FLBEIA output.
There are two types of functions:

6.1 Summary functions

Functions that summarise the results of the FLBEIA output in data frames.

`summary_flbeia:` An array with four dimensions: stock, year, iteration, indicator. The indicators are: recruitment, ssb, f, biomass, catch, landings and discards.

`B_flbeia:` Biomass values by stock. An array with three dimensions: stock, year and iteration.

`F_flbeia:` Fishing mortality values by stock. An array with three dimensions: stock, year and iteration.

`SSB_flbeia:` Spawning stock biomass values by stock. An array with three dimensions: stock, year and iteration.

`R_flbeia:` Recruitment values by stock. If the stock follows a biomass dynamics, then this function gives the growth. An array with three dimensions: stock, year and iteration.

`C_flbeia:` Catches by fleets and stock. An array with three dimensions: stock, year and iteration.

`L_flbeia:` Landings by fleets and stock. An array with three dimensions: stock, year and iteration.

`D_flbeia:` Discards by fleets and stock. An array with three dimensions: stock, year and iteration.

`advSum, advSumQ:` Data frame with the indicators related with the management advice (TAC). The indicators are: catch, discards, discRat, landings, quotaUpt and tac.

`bioSum, bioSumQ:` Data frame with the biological indicators. The indicators are: biomass, catch, catch.iyv, discards, disc.iyv, f, landings, land.iyv, rec and ssb.

`fltSum, fltSumQ:` Data frame with the indicators at fleet level. The indicators are: capacity, catch, costs, discards, discRat, effort, fcosts, gva, income, landings, netProfit, nVessels, price, profits, quotaUpt, salaries, vcosts and profitability.

`fltStkSum, fltStkSumQ:` Data frame with the indicators at fleet and stock level. The indicators are: landings, discards, catch, price, quotaUpt, tacshare, discRat and quota.

`npv:` A data frame with the net present value per fleet over the selected range of years.

`mtSum, mtSumQ:` Data frame with the indicators at fleet. The indicators are: effshare, effort, income and vcost.

`mtStkSum, mtStkSumQ:` Data frame with the indicators at fleet and metier level. The indicators are: catch, discards, discRat, landings and price.

`riskSum:` A data frame with the risk indicators. The indicators are: pBlim, pBpa and pPrflim.

`vesselSum, vesselSumQ:` Data frame with the indicators at vessel level. The indicators are: catch, costs, discards, discRat, effort, fcosts, gva, income, landings, netProfit, price, profits, quotaUpt, salaries, vcosts and profitability.

`vesselStkSum, vesselStkSumQ:` Data frame with the indicators at vessel and stock level. The indicators are: landings, discards, catch, price, quotaUpt, tacshare, discRat and quota.

`ecoSum_damara:` ecoSum built in the framework of Damara project.

`vesselStkSum, vesselStkSumQ:` Data frame with the indicators at vessel and stock level. The indicators are: landings, discards, catch, price, quotaUpt, tacshare, discRat and quota.

The data frames can be of wide or long format. In long format all the indicators are in the same column. There is one column, indicator, for the name of the indicator and a second one value for the numeric value of the indicator. In the wide format each of the indicators correspond with one column in the data frame. The long format it is recommendable to work with ggplot2 functions for example while the wide format it is more efficient for memory allocation and speed of computations.

The quantile version of the summaries, fooQ, returns the quantiles of the indicators. In the long format as many columns as elements in prob are created. The name of the columns are the elements in prob preceded by a q. In the wide format for each of the indicators as many columns as elements in prob are created. The names of the colums are the elements in prob preceded by q_name_of_the_indicator.

6.2 Plotting functions

Plotting functions to summarise the results of the FLBEIA output.

`plotFLBiols:` For each stock, returns a pdf with plots using `FLBiols` object. With plots on biomass in numbers at age, mean weight at age, fecundity, natural mortality, maturity, spawning, recruitment and spawning stock biomass.

`plotFLFleets:` For each fleet, returns a pdf with plots using `FLFleets` object. With plots on catch, discards, landings, capacity, crewshare, effort, fcost, effshare, and for each metier of this fleet: landings and discards at age in numbers and mean weight, alpha, beta and catch.q.

`plotCatchFl:` Returns a pdf with plots using `FLFleets` and `advice` objects. Whith plots on landings, discards and price by fleet.

`plotEco:` For each fleet, returns a pdf with plots using `FLFleets` object. With plots on capacity, costs, effort, profits by fleet.

6.3 Functions for joining iterations

`joinIter` function allows to join iterations of an `FLBEIA` output object with one unique iteration. This is very usefull in case that the different iterations are run separately in a cluster and we want to join them afterwards.

The call to `joinIter` function within `FLBEIA` is done as:

```
joinIter(object, files, directory, Niters, elements, advice.ext = , fleets.ctrl.ext =
```

Arguments description:

`object:` Character. Corresponds to the name of the object that must be 'joined'. This object must be the output of BEIA function.

`files:` Character vector with the names of the files. It has to be noted that each of the files must contain a single object.

`directory:` The directory where the files are stored. The default is the current directory.

`Niters:` Numeric vector with the number of iterations per object. If $length = 1$, then it is assumed that all objects have the same number of iterations.

`elements:` The elements of the objects that must be joined. The default is to join all the elements.

`advice.ext:` The type of advice that is given. Options are: `'TAC'` (default value) or `'TAE'`.

`fleets.ctrl.ext:` Character. Name of the object in `fleets.ctrl` that needs iterations to be joined. Default is `'seasonal.share'`.

Another alternative to join iterations is calculating summary statistics for each iteration and afterwards joining the summary results.

++++ ver mail Leire Ibaibarriaga ++++

7 Acknowledgements

This work was carried out with financial support from Basque Country Government (Eusko Jaurlaritza - Gobierno Vasco, Dirección General de Pesca, Viceconsejería de Agricultura, Pesca y Politicas Alimentarias - Departamento de Desarrollo Económico y Competitividad) and the European Commission under the DEEPFISHMAN project (Grant agreement no. 979227390), Myfish project (Grant agreement no. 289257), FACTS project (Grant agreement no. 244966) and by the Directorate-General for Maritime Affairs and Fisheries (Request for services - Commitment no. SI2.676322).

We would also like to thank `R` and `FLR` teams for their work in the code development for the R system.

A New FLR - S4 classes

A.1 FLBDsim class

FLBDsim class has been created in order to facilitate the simulation of population growth in populations aggregated in biomass, i.e. $g(.)$ in Equation **??**. The population dynamics are simulated as follows:

$$B_{y,s} = B_{y0,s0} + g(B_{y0,s0}) \cdot \varepsilon_{y,s} - C_{y0,s0} \tag{33}$$

where B is the biomass, C the catch, $y0$ and $s0$ are the subscripts of previous season's year and season and ε is the uncertainty value in year y and season s. It is a S4 class and has 10 slots:

name, desc, range: Slots common to all FLR objects.

model: Character string or formula. If character, it must coincide with an already existing growth model. If formula, the parameters must be slots in the object or elements of covar slot. Currently, there is only one growth model available, 'PellaTom' that corresponds with Pella-Tomlinson growth model (**?**).

biomass: FLQuant to store biomass in weight. The dimension in quant, unit and area must be equal to 1 and in the rest of the dimensions it must be congruent with general simulation settings.

catch: FLQuant to store total catch in weight. The dimension in quant, unit and area must be equal to 1 and in the rest of the dimensions it must be congruent with general simulation settings.

uncertainty: FLQuant to store the error that is multiplied to the point estimate of growth . The dimension in quant, unit and area must be equal to 1 and in the rest of the dimensions it must be congruent with general simulation settings. Thus, a different error can be used for each year, season and iteration.

params: An array to store the parameters of the model. The dimensions of the array are params, year, season, iter. The dimension in year, season and iter must be congruent with general simulation settings. Thus, a different set of parameters can be used for each year, season and iteration.

covar: An FLQuants object. The elements of the list are used to store covariates' values and it is used to apply growth models with covariates. Its functionality is the same as in FLSR object.

alpha: An array with dimension [year = ny, season = ns, iteration = ni] with year, season and iteration dependent value bigger than one which indicates, in percentage, how big can be the biomass in comparison with the carrying capacity.

A.2 FLSRSim class

FLSRsim class has been created in order to facilitate the simulation of recruitment in age structured populations. The recruitment dynamics are simulated as follows:

$$R_{y,s} = \Phi(S_{y-tl_0,s-tl_1}, covars_{y-tl_0,s-tl_1}) \cdot \varepsilon_{y,s} \cdot \rho_{y,s} \tag{34}$$

where $R_{y,s}$ is the recruitment in year y and season s, Φ is the stock-recruitment model, tl_0 and tl_1 are the year and season lag between spawning and recruitment, respectively, S_{y-tl_0,tl_1} and $covars_{y-tl_0,s-tl_1}$ are the stock index and covariates in year $y - tl_0$ and season tl_1, $\varepsilon_{y,s}$ is the uncertainty value in year y and season s and $\rho_{y,s}$ is the proportion of recruitment that recruits in year y and season s and is produced by stock index S in year $y - tl_0$ and season tl_1 .

name, desc, range: Slots common to all FLR objects.

rec: An FLQuant with dimension $[1, ny, 1, ns, 1, ni]$ used to store recruitment.

ssb: An FLQuant with dimension $[1, ny, 1, ns, 1, ni]$ used to store SSB or the stock index used in the stock-recruitment relationship.

covar: An FLQuants to store the covariates used in the stock-recruitment relationship. For details on the use of this slot look at the description of FLSR class.

uncertainty: An FLQuant with dimension $[1, ny, 1, ns, 1, ni]$ used to store the uncertainty related to stock-recruitment process. The content of this slot is multiplied to the point estimate of recruitment. As its effect is multiplicative, then set it equal to 1 for all year, season and iteration if uncertainty is not going to be considered around stock-recruitment curve.

`proportion:` An `FLQuant` with dimension $[1, ny, 1, ns, 1, ni]$ used to store the proportion of the recruitment produced by stock index in year $y -$ `timelag`$[1, s]$ and season `timelag`$[2, s]$ that recruits in year y and season s.
The content of this slot is multiplied to the point estimate of recruitment. As its effect is multiplicative, then set it equal to 1 if all the recruitment produced by certain stock index is recruited at the same time and set it equal to 0 if none of the recruitment produced by certain stock index is recruited in that season.

`model:` `Character string` or `formula`. If character it specifies the name of the function used to simulate the recruitment. If formula the left hand side of $\sim$ must be equal to `rec` and the elements in right hand side must be among `ssb, covars` and `params`.

`params:` An array with dimension `[nparams, ny, ns, ni]`, thus, the parameters may be year, season and iteration dependent. Year dimension in parameters may be useful to model regime shifts.

`timelag:` A matrix with dimension `[2,ns]`. This object indicates the time lag between spawning and recruitment in each season. For each season, the element in the first row indicates the age at recruitment and the element in the second row indicates the season at which the recruitment was spawn.

B Graphical representation of FLR objects

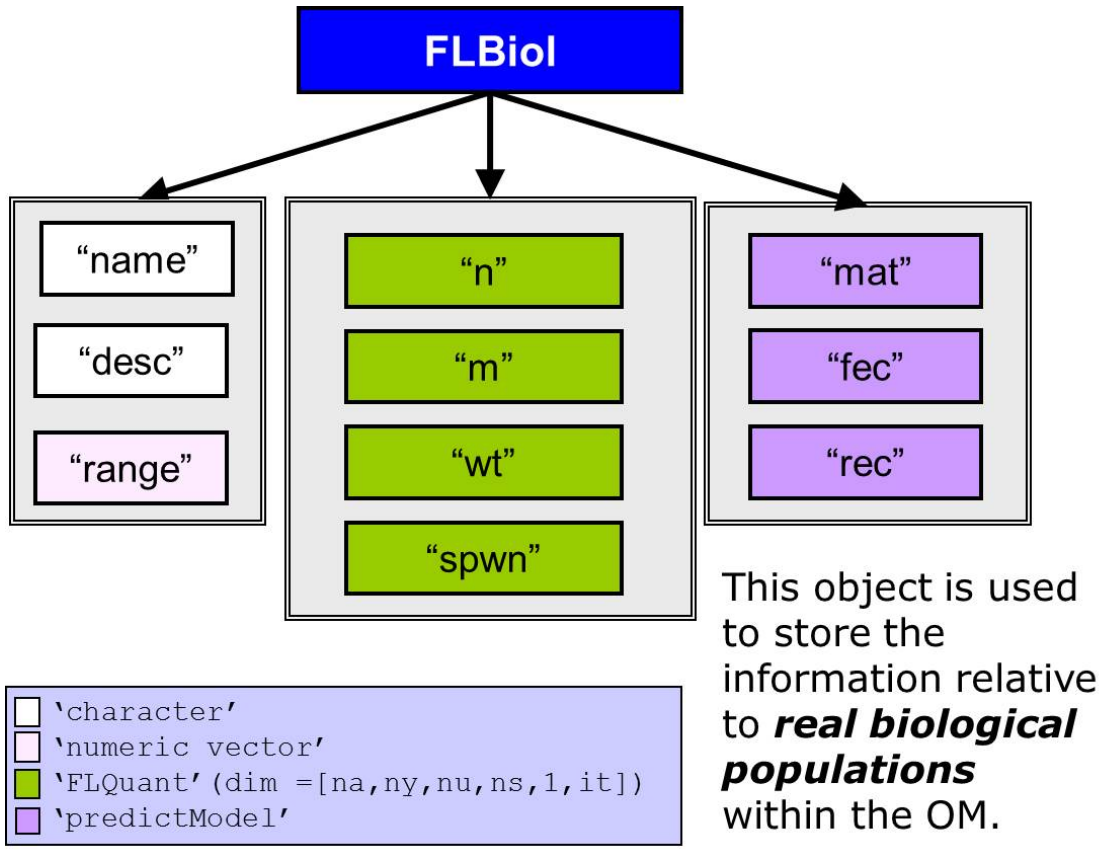

Figure B.1: FLBiol object

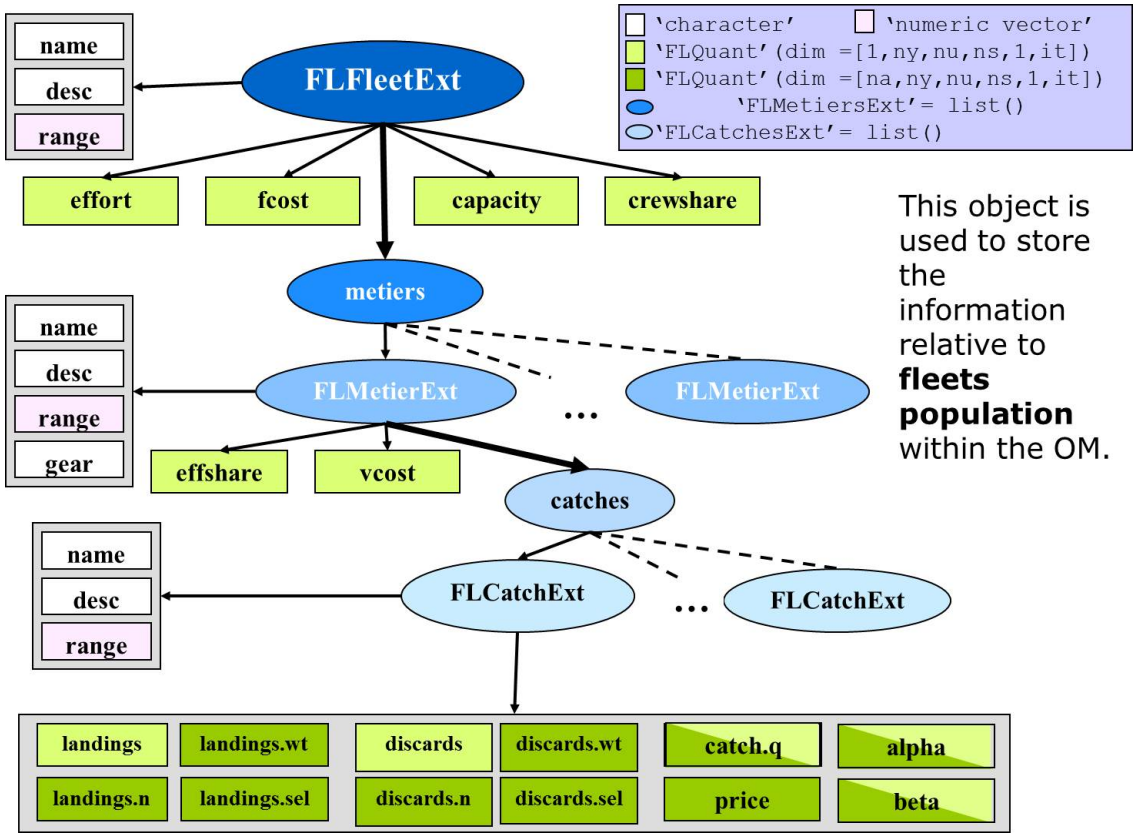

Figure B.2: FLFleetExt object

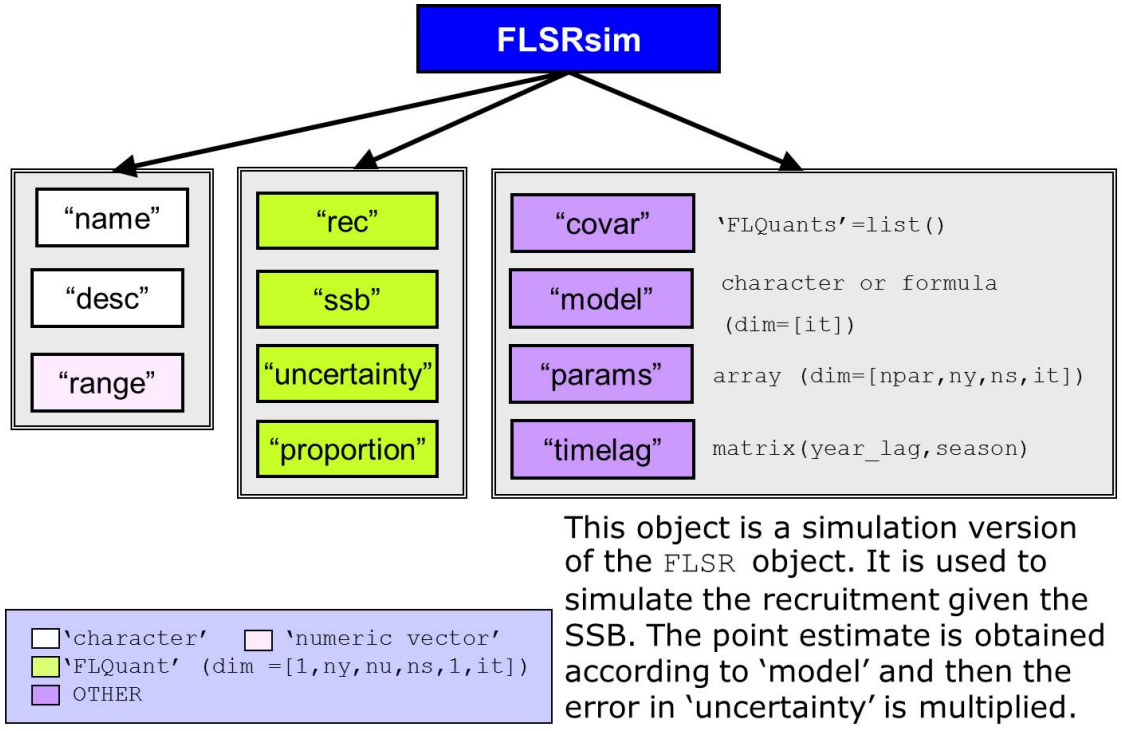

Figure B.3: FLSRsim object

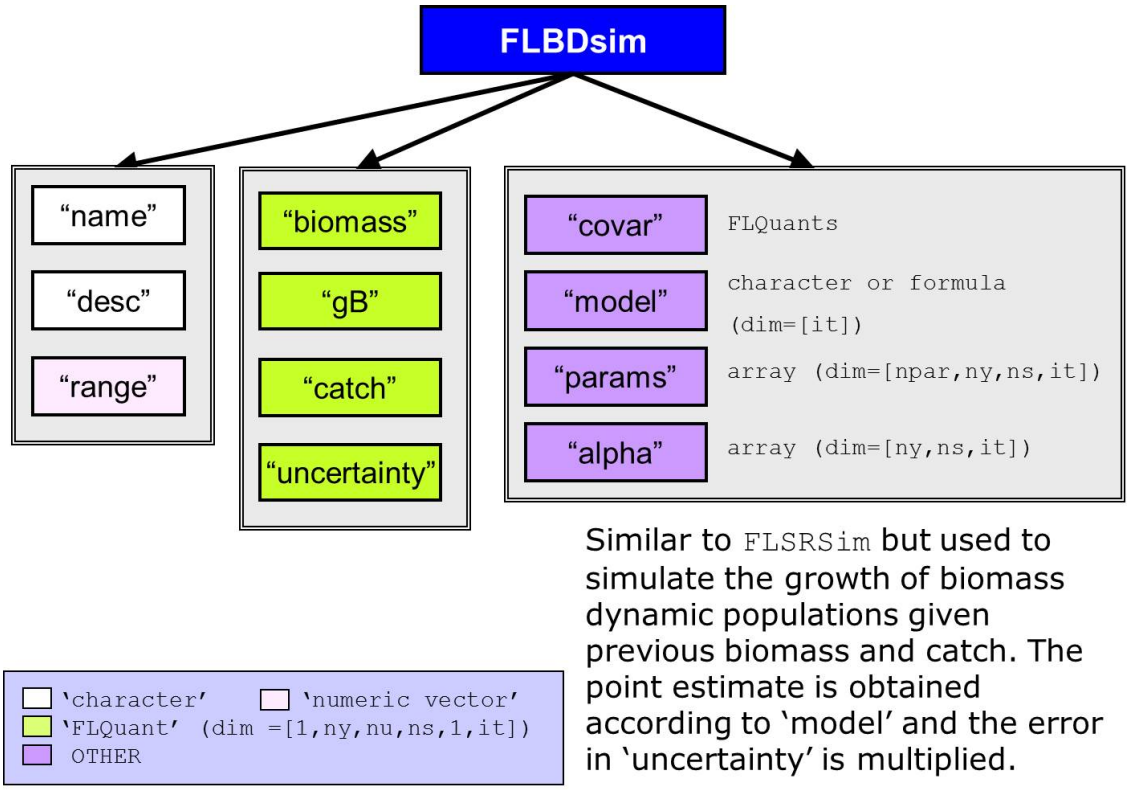

Figure B.4: FLBDsim object

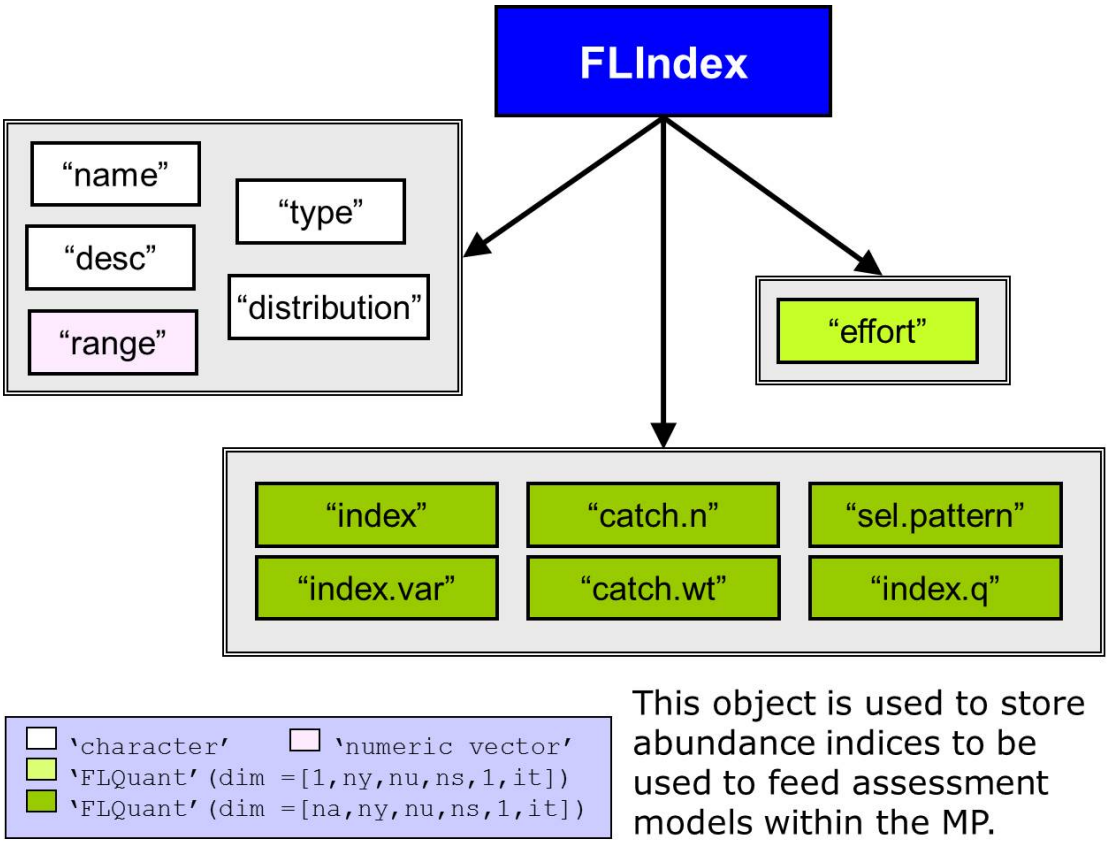

Figure B.5: FLIndex object

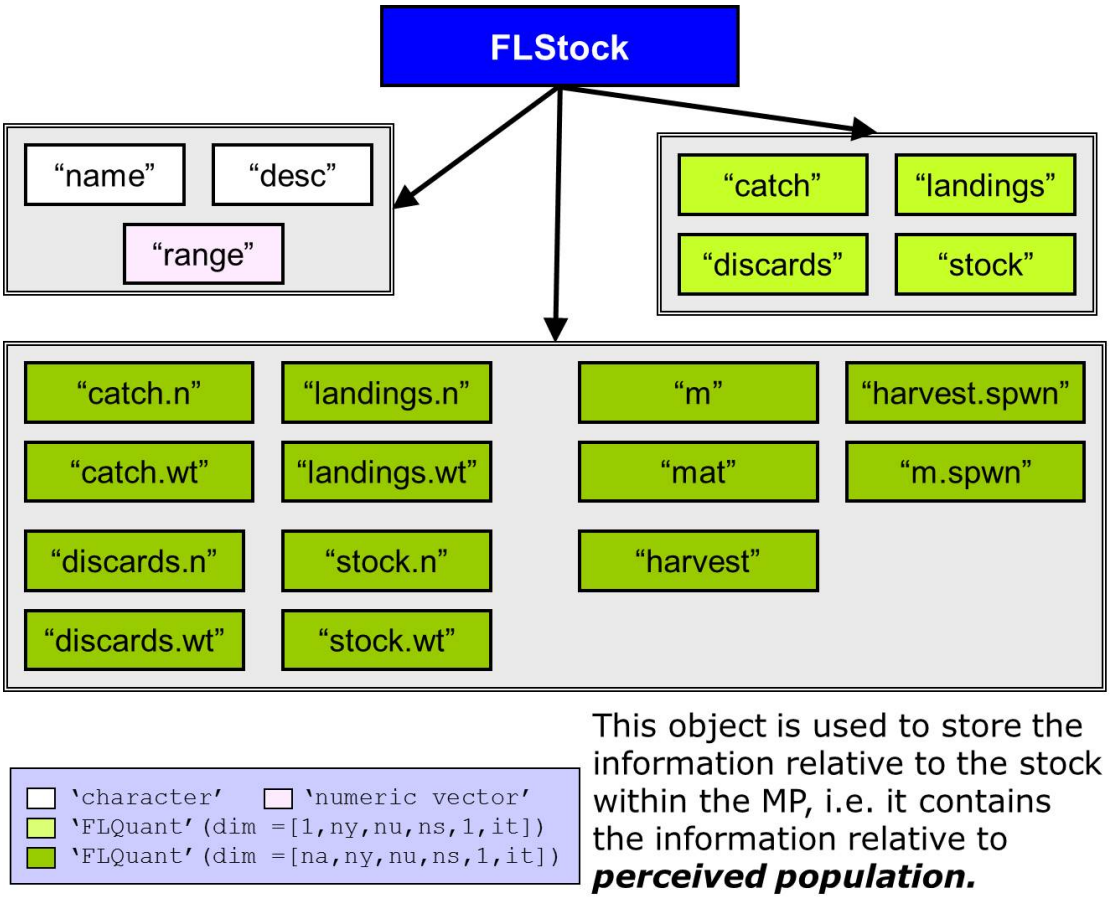

Figure B.6: FLStock object

C Graphical representation of control objects

Table C.1: Description of all the optional arguments for `main.ctrl` object (of class list). The arguments with * are compulsory arguments.

Argument	class	Dimension	Values	Required for
sim.years*	numeric vector	2 (initial,final)	any in year range	

Table C.2: Description of all the optional arguments for `biols.ctrl` object (of class list). The arguments with * are compulsory arguments.

Argument	class	Dimension	Values	Required for
[[st]]$growth.model*	character	1	`'fixedPopulation','ASPG'`	`BDPG`

Table C.3: Description of all the optional arguments for `fleets.ctrl` object (of class list). In the table we assume that `stk` is the name of the stock and `fl` the name of the fleet. The arguments with * are compulsory arguments.

Argument	class	Dimension	Values	Required for
catch.treshold	FLQuant	[nst,ny,1,ns,1,ni]	Proportions in [0,1] range	`SMFB`, `SSFB`
seasonal.share[[st]]	FLQuant	[nst,ny,1,ns,1,ni]	Proportionsin [0,1] (sum along seasons = 1)	`SMFB`, `SSFB`
[[fl]]$effort.model*	character	1	`'fixedEffort'`,`'SMFB'`,`'SSFB'`, `'MaxProfit'`, `'MaxProfitSeq'`	
[[fl]]$restriction	character	1	`'catch'`,`'landings'`	`SMFB`, `SSFB`
[[fl]]$effort.rest	character	1	`'max'`,`'min'`,`'mean'`,`'prev'`,`stock.name`	`SMFB`, `SSFB`
[[fl]]$effectiveDay.perc	FLQuant	[1,ny,1,ns,1,ni]	Proportions in [0,1]	`SSFB`
[[fl]]$effort.realloc	character	1	`NULL`,`'curr.eff'`	`SSFB`
[[fl]]$stk.cnst	character	1	`stock.name`	`MaxProfit`
[[fl]]$capital.model*	character	1	`'fixedCapital'`,`'SCD'`	
[[fl]][[st]]$catch.model*	character	1	`'cobbDouglasBio'`,`'cobbDouglasAge'`, `'seasonshare'`	
[[fl]][[st]]$catch.dependence[1]	character	nfl	`fleet.name`	`seasonShare`
[[fl]][[st]]$TAC.OS.model	character	1	`'TAC.OS.triangCond'`	`SMFB`, `MaxProfit`
[[fl]][[st]]$TAC.OS.triangCond.params	named numeric vector	3 (min,max,mode)		`SMFB`, `MaxProfit`
[[fl]][[st]]$discard.TAC.OS	logical	1	`'TRUE'` (TAC overshoot is discarded), `'FALSE'` (TAC overshoot is included in landings)	`SMFB`, `MaxProfit`
[[fl]][[st]]$price.model*	character	1	`'fixedPrice'`, `'elasticPrice'`	
[[fl]][[st]]$pd.els	numeric array	[na,ns,ni]		`elasticPrice`
[[fl]][[st]]$pd.La0	numeric array	[na,ns,ni]		`elasticPrice`
[[fl]][[st]]$pd.Pa0	numeric array	[na,ns,ni]		`elasticPrice`
[[fl]][[st]]$pd.total	logical	1	`'TRUE'` (if depending on total catch), `'FALSE'`	`elasticPrice`

nst: number of stocks
nfl: number of fleets
na: number of age clases
ny: number of years
ns: number of seasons
ni: number of iterations
[1]If defined, takes the same seasonal share as the one of fleet `fleet.name`.

Table C.4: Description of all the optional arguments for `covars.ctrl` object (of class list). In the table we assume that `cv` is the name of the covariate. The arguments with * are compulsory arguments.

Argument	class	Dimension	Values	Required for
[[cv]]$process.model*	character	1	`'fixedCovar'`,`'ssb.get'`	
[[cv]]$ssb.stock	character	1	`stock.name`	`ssb.get`
[[cv]]$spwn.season	numeric	1	`season`	`ssb.get`
[[cv]]$sr.covar	character	1	`stock.name`	`ssb.get`

Table C.5: Description of all the optional arguments for `obs.ctrl` object (of class list). In the table we assume that `stk` is the name of the stock and `id` the name of the index. The arguments with * are compulsory arguments.

+++++ SONIA: no descrito nada de esto en el texto, necesitariamos extender informacion?

Argument	class	Dimension	Values	Required for
[[st]]$stkObs$stkObs.model*	character	1	'NoObsStock','perfectObs', 'age2ageDat','age2agePop', 'age2bioDat','age2bioPop','bio2bioDat','bio2bioPop'	
[[st]]$stkObs$TAC.ovrsht	array	[1,ny]	In percentage per unit	age2ageDat,age2bioDat,bio2bioDat
[[st]]$stkObs$ages.error[2]	array	[na,na,ny,ni]	In percentage per unit	age2ageDat,age2agePop
[[st]]$stkObs$nmort.error	FLQuant	[na,ny,1,1,1,ni]	any	age2ageDat
[[st]]$stkObs$mat.error	FLQuant	[na,ny,1,1,1,ni]	any	age2ageDat
[[st]]$stkObs$stk.nage.error	FLQuant	[na,ny,1,1,1,ni]	any	age2agePop
[[st]]$stkObs$stk.wgt.error	FLQuant	[na,ny,1,1,1,ni]	any	age2agePop
[[st]]$stkObs$stk.bio.error	FLQuant	[1,ny,1,1,1,ni]	any	age2bioPop,bio2bioDat,bio2bioPop
[[st]]$stkObs$land.nage.error	FLQuant	[na,ny,1,1,1,ni]	any	age2ageDat
[[st]]$stkObs$land.wgt.error	FLQuant	[na,ny,1,1,1,ni]	any	age2ageDat
[[st]]$stkObs$land.bio.error	FLQuant	[1,ny,1,1,1,ni]	any	age2bioDat,bio2bioDat,bio2bioPop
[[st]]$stkObs$disc.nage.error	FLQuant	[na,ny,1,1,1,ni]	any	age2ageDat
[[st]]$stkObs$disc.wgt.error	FLQuant	[na,ny,1,1,1,ni]	any	age2ageDat
[[st]]$stkObs$disc.bio.error	FLQuant	[1,ny,1,1,1,ni]	any	age2bioDat,bio2bioDat,bio2bioPop
[[st]]$indObs[[id]]$indObs.model*	character	1	'NoObsIndex','NoObservation','ageInd', 'bioInd'	

na: number of age clases
ny: number of years
ni: number of iterations
[2]For each year and iteration, there is a square matrix whose column elements quantify the probability that a fish of age a is classified as having any age between $min(age)$ and $max(age)$.

Table C.6: Description of all the optional arguments for `assess.ctrl` object (of class list). In the table we assume that `stk` is the name of the stock. The arguments with * are compulsory arguments.

+++++ SONIA: no descrito en el texto, necesitariamos extender informacion?

Argument	class	Dimension	Values	Required for
[[st]]$assess.model*	character	1	'NoAssessment','FLXSAnew',...	
[[st]]$control	control object		Depends on the selected assessment model (e.g. `FLXSA.control()` for XSA assessment)	`assess.ctrl[[st]]$assess.model`

Table C.7: Description of all the optional arguments for `advice.ctrl` object (of class list). In the table we assume that `stk` is the name of the stock and `id` the name of the index. The arguments with * are compulsory arguments.

Argument	class	Dimension	Values	Required for
[[st]]$HCR.model*	character	1	'fixedAdvice','annualTAC','IcesHCR','FroeseHCR', 'annexIVHCR','ghlHCR','aneHCRE','neaMAC_ltmp', 'F2CatchHCR','little2011HCR','pidHCR','pidHCRtarg', 'MAPHCR','CFPMSYHCR','MultiStockHCR'	
[[st]]$AdvCatch	logical	1	'TRUE' (TAC in terms of catch), 'FALSE' (TAC in terms of landings)	annualTAC, IcesHCR, CFPMSYHCR, MAPHCR, F2CatchHCR
[[st]]$nyears	numeric	1	season.name	annualTAC, IcesHCR, F2CatchHCR, MultiStockHCR
[[st]]$wts.nyears	numeric	1	season.name	annualTAC, IcesHCR, MAPHCR, CFPMSYHCR, F2CatchHCR, MultiStockHCR
[[st]]$fbar.nyears	numeric	1	season.name	annualTAC, IcesHCR, MAPHCR, CFPMSYHCR, F2CatchHCR, MultiStockHCR
[[st]]$f.rescale	logical	1	'TRUE' (???), 'FALSE'	annualTAC, IcesHCR, MAPHCR, CFPMSYHCR, F2CatchHCR, MultiStockHCR
[[st]]$disc.nyears	numeric	1		annualTAC, CFPMSYHCR
[[st]]$fwd.control	fwdBDcontrol			annualTAC
[[st]]$sr*model*	character	1	SR model name	annualTAC, IcesHCR, MAPHCR, F2CatchHCR, MultiStockHCR
[[st]]$sr*params*[1]	FLPar	[npar,ni]		annualTAC, IcesHCR, MAPHCR, CFPMSYHCR, F2CatchHCR, MultiStockHCR
[[st]]$sr*years*[1]	named numeric vector	2 (y.rm, num.years)		annualTAC, IcesHCR, MAPHCR, CFPMSYHCR, F2CatchHCR, MultiStockHCR
[[st]]$growth.years[2]	named numeric vector	2 (y.rm, num.years)		annualTAC, IcesHCR, CFPMSYHCR, F2CatchHCR, MultiStockHCR
[[st]]$ref.pts	matrix	[nrp,ni]		IcesHCR, FroeseHCR, annexIVHCR, ghlHCR, MAPHCR, CFPMSYHCR, F2CatchHCR, little2011HCR, pidHCR, pidHCRtarg, MultiStockHCR
[[st]]$intermediate.year	character	1	'Fsq' or any	IcesHCR, neaMAC_ltmp, F2CatchHCR, MultiStockHCR
[[st]]$index	character	1	index.name	annexIVHCR, little2011HCR, pidHCR, pidHCRtarg
[[st]]$type	numeric	1	2 or 4	annexIVHCR
[[st]]$N	numeric	1		MAPHCR,CFPMSYHCR
[[st]]$stocksInHCR	vector	any	names of the stocks to be taken into account	MultiStockHCR

npar: number of parameters in the SR model selected for stock `st`.
nrp: number of reference points required for the HCR selected for stock `st`.
ny: number of years
ni: number of iterations
[1]Optional argument for the function.
[2]Used only for stocks aggregated in biomass.

D Smart conditioning - function's arguments description

Table D.1: Description of the arguments of the function `create.biols.data`. In the table we assume that `stk` is the name of the stock. All the arguments are required.

Argument		class	Dimension	Definition
yrs		vector	3	c(first.yr, proj.yr, last.yr)
	first.yr	numeric	1	First year of simulation
	proj.yr	numeric	1	First year of projection
	last.yr	numeric	1	Last year of projection
ni		numeric	1	Number of iterations
ns		numeric	1	Number of seasons
stks.data		list	number of stocks	List with the name of the stocks and the following elements:
	stk.unit	numeric	1	Number of units
	stk.age.min	numeric	1	Minimum age
	stk.age.max	numeric	1	Maximum age
	stk_n.flq	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Abundance in numbers at age
	stk_wt.flq	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Weight at age
	stk_m.flq	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Natural mortality mortality rate
	stk_fec.flq	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Fecundity
	stk_mat.flq	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Percentage of mature individuals
	stk_spwn.flq	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Proportion of time step at spawning
	stk_range.min	numeric	1	Minimum age
	stk_range.max	numeric	1	Maximum age
	stk_range.plusgroup	numeric	1	Plusgroup age
	stk_range.minyear	numeric	1	Minimum year
	stk_range.maxyear	numeric	1	Maximum year
	stk_range.minfbar	numeric	1	Minimum age for calculating average fishing mortality
	stk_range.maxfbar	numeric	1	Maximum age for calculating average fishing mortality
	stk_biol.proj.avg.yrs	vector	any	Historic years to calculate averages (in spwn, fec, m and wt) for the projection period

na: number of age (from min.age to max.age)
ny(hist): number of historic years (from first.yr to proj.yr-1)
1/nu(stock): 1 or number of units of the stock
1/ns: 1 or number of seasons
1/ni: 1 or number of iterations

Table D.2: Description of the arguments of the function `create.SRs.data`. In the table we assume that `stk` is the name of the stock. The arguments with superscript * are optional arguments.

Argument		class	Dimension	Definition
yrs		vector	3	c(first.yr, proj.yr, last.yr)
	first.yr	numeric	1	First year of simulation
	proj.yr	numeric	1	First year of projection
	last.yr	numeric	1	Last year of projection
ni		numeric	1	Number of iterations
ns		numeric	1	Number of seasons
stks.data		list	number of stocks	List with the name of the stocks and the following elements:
	stk.unit	numeric	1	Number of units
	stk.age	numeric	1	Number of age classes
	stk_sr.model	character	1	Name of the SR model
	stk_params.n	vector	1	Number of parameters
	stk_params.name	vector	stk_params.n	Name of the parameters
	stk_params.array	array	[stk_params.n,ny,ns,1/ni]	Parameter values
	stk_rec.flq	FLQuant	[1,ny(hist),1/nu(stock),1/ns,1/ni]	Recruitment values
	stk_ssb.flq	FLQuant	[1,ny(hist),1/nu(stock),1/ns,1/ni]	Spawning stock values
	stk_uncertainty.flq*	FLQuant	[1,ny,1/nu(stock),1/ns,1/ni]	Uncertainty
	stk_proportion.flq	FLQuant	[1,ny,1/nu(stock),1/ns,1/ni]	Recruitment distribution in each time step. For details see `FLSRsim`
	stk_prop.avg.yrs	vector	any	Historical years to calculate the proportion average
	stk_timelag.matrix	matrix	(2,ns)	Timelag between the spawning an recruitment (time.lag.yr, time.lag.ns) For details see `FLSRsim`
	stk_range.min	numeric	1	Minimum age
	stk_range.max	numeric	1	Maximum age
	stk_range.plusgroup	numeric	1	Plusgroup age
	stk_range.minyear	numeric	1	Minimum year

na: number of age (from min.age to max.age)
ny(hist): number of historic years (from first.yr to proj.yr-1)
ny: number of years (from first.yr to last.yr)
1/nu(stock): 1 or number of units of the stock
1/ns: 1 or number of seasons
1/ni: 1 or number of iterations
ns: number of seasons

Table D.3: Description of the arguments of the function `create.BDs.data`. In the table we assume that `stk` is the name of the stock. The arguments with * are optional arguments.

Argument		class	Dimension	Definition
yrs		vector	3	c(first.yr, proj.yr, last.yr)
	first.yr	numeric	1	First year of simulation
	proj.yr	numeric	1	First year of projection
	last.yr	numeric	1	Last year of projection
ni		numeric	1	Number of iterations
ns		numeric	1	Number of seasons
stks.data		list	number of stocks	List with the name of the stocks and the following elements:
	stk.unit	numeric	1	Number of units
	stk_bd.model	character	1	Name of the BD model
	stk_params.name	vector	np	Name of the parameters
	stk_params.array	vector	np	Parameter values
	stk_biomass.flq	FLQuant	[1,ny(hist),1/nu(stock),1/ns,1/ni]	Biomass values
	stk_catch.flq	FLQuant	[1,ny(hist),1/nu(stock),1/ns,1/ni]	Catch values
	stk_range.min	numeric	1	Minimum age
	stk_range.max	numeric	1	Maximum age
	stk_range.plusgroup	numeric	1	Plusgroup age
	stk_range.minyear	numeric	1	Minimum year
	stk_alpha	numeric	1	Maximum variability of carrying capacity
	stk_gB.flq*	FLQuant	[1,ny(hist),1/nu(stock),1/ns,1/ni]	Surplus production values
	stk_uncertainty.flq*	FLQuant	[1,ny,1/nu(stock),1/ns,1/ni]	Uncertainty

na: number of age (from min.age to max.age)
ny(hist): number of historic years (from first.yr to proj.yr-1)
ny: number of years (from first.yr to last.yr)
1/nu(stock): 1 or number of units of the stock
1/ns: 1 or number of seasons
1/ni: 1 or number of iterations
np: number of parameters in BD model

Table D.4: Description of the arguments of the function `create.fleets.data`. In the table we assume that `stk` is the name of the stock,`fl` the name of the fleet and `met` the name of the metier. The arguments with * are optional arguments.

Argument		class	Dimension	Definition
yrs		vector	3	c(first.yr, proj.yr, last.yr)
	first.yr	numeric	1	First year of simulation
	proj.yr	numeric	1	First year of projection
	last.yr	numeric	1	Last year of projection
ni		numeric	1	Number of iterations
ns		numeric	1	Number of seasons
fls.data		list	number of fleets	List with the name of the fleets and the following elements:
	fl.met	vector	number of metiers in 'fl'	Name of the metiers in the fleet 'fl'
	fl.met.stks	vector	number of stocks in 'fl.met'	Name of the stocks in the metier 'met' and fleet 'fl'
	fl_effort.flq	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Effort for 'fl' fleet
	fl_capacity.flq*	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Capacity of 'fl' fleet
	fl_fcost.flq*	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Fixed costs for 'fl' fleet
	fl_crewshare.flq*	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Crewshare for 'fl' fleet
	fl.met_effshare.flq	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Effort share for fl' fleet and 'met' metier
	fl.met_vcost.flq*	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Variable costs for 'fl' fleet and 'met' metier
	fl.met.stk_landings.n.flq	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Landings in numbers at age for fl' fleet,'met' metier and 'stk' stock
	fl.met.stk_landings.wt.flq*	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Mean weight of landings at age for 'fl' fleet and 'met' metier
	fl.met.stk_discards.n.flq*	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Discards in numbers at age for 'fl' fleet and 'met' metier
	fl.met.stk_discards.wt.flq*	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Mean weight at age in discards for 'fl' fleet and 'met' metier
	fl.met.stk_price.flq*	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Price at age for 'stk' stock in 'fl' fleet and 'met' metier
	fl.met.stk_alpha.flq*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Cobb-Douglas alpha parameter for 'fl' fleet, 'met' metier and 'stk' stock
	fl.met.stk_beta.flq*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Cobb-Douglas beta parameter for 'fl' fleet, 'met' metier and 'stk' stock
	fl.met.stk_catch.q.flq*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Cobb-Douglas catch.q parameter for 'fl' fleet, 'met' metier and 'stk' stock
	fl_proj.avg.yrs	vector	any	Historic years to calculate averages (in effort, fcost, crewshare, and capacity) in 'fl' fleet for the projection period
	fl.met_proj.avg.yrs*	vector	any	Historic years to calculate averages (in effshare and vcost) in 'fl' fleet and 'met' metier for the projection period
	fl.met.stk_proj.avg.yrs*	vector	any	Historic years to calculate averages (in landings.wt, discards.wt, landings.sel, discards.sel, alpha, beta and catch.q) in 'fl' fleet, 'met' metier and 'stk' stock for the projection period
stks.data		list	number of stocks	List with the name of the stocks and the following elements:
	stk.unit	numeric	1	Number of units
	stk.age	numeric	1	Number of age clases
	stk.age.min	numeric	1	Minimum age
	stk.age.max	numeric	1	Maximum age
	stk_wt.flq*	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Weight at age. Only required if fl.met.stk_landings.wt is not defined
	stk_n.flq*	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Numbers at age in the population (for stocks modelled in numbers at age). Only required if Cobb-Douglas parameters are not defined
	stk_gB.flq*	FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Biomass growth (for stocks modelled in biomass). Only required if Cobb-Douglas parameters are not defined

na: number of age (from min.age to max.age)
ny(hist): number of historic years (from first.yr to proj.yr-1)
ny: number of years (from first.yr to last.yr)
1/nu(stock): 1 or number of units of the stock
1/ns: 1 or number of seasons
1/ni: 1 or number of iterations

Table D.5: Description of the arguments of the function `create.indices.data`. In the table we assume that `stk` is the name of the stock and `ind` the name of the index. The arguments with * are optional arguments.

Argument		class	Dimension	Definition
yrs		vector	3	c(first.yr, proj.yr, last.yr)
	first.yr	numeric	1	First year of simulation
	proj.yr	numeric	1	First year of projection
	last.yr	numeric	1	Last year of projection
ni		numeric	1	Number of iterations
ns		numeric	1	Number of seasons
stks.data		list	number of stocks	List with the name of the stocks with indices and the following elements:
	stk.unit	numeric	1	Number of units
	stk.age	numeric	1	Number of age classess
	stk_indices	character	1	Name of indices for the stock 'stk'
	stk_ind_type*	character	1	Type of index
	stk_ind_distribution*	character	1	Name of the statistical distribution of the 'ind' index values for stock 'stk'
	stk_ind_index.flq	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Historical index data for index 'ind' of stock 'stk'
	stk_ind_index.var.flq*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Variability in 'ind' index of stock 'stk'
	stk_ind_index.q.flq*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Catchability for 'ind' index of stock 'stk'
	stk_ind_catch.n.flq*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Catch at age in numbers for 'ind' index of stock 'stk'
	stk_ind_catch.wt.flq*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Mean weight at age in the catch for 'ind' index of stock 'stk'
	stk_ind_effort.flq*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Effort for 'ind' index of stock 'stk'
	stk_ind_sel.pattern.flq*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Selection pattern for 'ind' index of stock 'stk'
	stk_ind_range.min*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Minimum age in 'ind' index of stock 'stk'
	stk_ind_range.max*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Maximum age in 'ind' index of stock 'stk'
	stk_ind_range.plusgroup*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Plusgroup age in 'ind' index of stock 'stk'
	stk_ind_range.minyear*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	First year with 'ind' index data of stock 'stk'
	stk_ind_range.maxyear*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Last year with 'ind' index data of stock 'stk'
	stk_ind_range.startf*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Minimum age for calculating average fishing mortality for 'ind' index of stock 'stk'
	stk_ind_range.endf*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Maximum age for calculating average fishing mortality for 'ind' index of stock 'stk'

na: number of age (from min.age to max.age)
ny: number of years (from first.yr to last.yr)
1/nu(stock): 1 or number of units of the stock
1/ns: 1 or number of seasons
1/ni: 1 or number of iterations

Table D.6: Description of the arguments of the function `create.advice.data`. In the table we assume that `stk` is the name of the stock. The arguments with * are optional arguments.

Argument		class	Dimension	Definition
yrs		vector	3	c(first.yr, proj.yr, last.yr)
	first.yr	numeric	1	First year of simulation
	proj.yr	numeric	1	First year of projection
	last.yr	numeric	1	Last year of projection
ni		numeric	1	Number of iterations
ns		numeric	1	Number of seasons
stks.data		list	number of stocks	List with the name of the stocks with indices and the following elements:
	stk_advice.TAC.flq*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	TAC of the stock 'stk'
	stk_advice.TAE.flq*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	TAE of the stock 'stk'
	stk_advice.quota.share.flq*	FLQuant	[na,ny,1/nu(stock),1/ns,1/ni]	Quota share of the stock 'stk'
	stk_advice.avg.yrs*	FLQuant	any	Mean weight at age in the catch for 'ind' index of stock 'stk'
fleets*		FLQuant		Only required if `stk_advice.quota.share` is not specified. Can be the output of `create_fleets_FLBEIA` function

na: number of age (from min.age to max.age)
ny: number of years (from first.yr to last.yr)
1/nu(stock): 1 or number of units of the stock
1/ns: 1 or number of seasons
1/ni: 1 or number of iterations

Table D.7: Description of the arguments of the function `create.list.stks.flqa`. In the table we assume that `stk` is the name of the stock.

Argument		class	Dimension	Definition
stks		vector	number of stocks	Name of all the stocks
yrs		vector	3	c(first.yr, proj.yr, last.yr)
	first.yr	numeric	1	First year of simulation
	proj.yr	numeric	1	First year of projection
	last.yr	numeric	1	Last year of projection
ni		numeric	1	Number of iterations
ns		numeric	1	Number of seasons
list.stks.unit		list	number of stocks	List with the name of the stocks and each stock contains the number of units
list.stks.age		list	number of stocks	List with the name of the stocks and each stock contains a vector with minimum age (min.age) and maximum age (max.age)

Table D.8: Description of the arguments of the function `create.list.stks.flq`. In the table we assume that `stk` is the name of the stock.

Argument		class	Dimension	Definition
stks		vector	number of stocks	Name of all the stocks
yrs		vector	3	c(first.yr, proj.yr, last.yr)
	first.yr	numeric	1	First year of simulation
	proj.yr	numeric	1	First year of projection
	last.yr	numeric	1	Last year of projection
ni		numeric	1	Number of iterations
ns		numeric	1	Number of seasons
list.stks.unit		list	number of stocks	List with the name of the stocks and each stock contains the number of units

Table D.9: Description of the arguments of the function `calculate.CDparam`. In the table we assume that `stk` is the name of the stock. The arguments with * are optional arguments.

Argument	class	Dimension	Definition	
stk.n		FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Abundance in numbers at age
landings.n		FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Landings in numbers at age
discards.n		FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	Discards in numbers at age
effort		FLQuant	[1,ny(hist),1/nu(stock),1/ns,1/ni]	Effort
effshare		FLQuant	[1,ny(hist),1/nu(stock),1/ns,1/ni]	Effort share
age.min		numeric	1	Minimum age
age.max		numeric	1	Maximum age
flqa		FLQuant	[na,ny(hist),1/nu(stock),1/ns,1/ni]	An FLQuant object
flq		FLQuant	[1,ny(hist),1/nu(stock),1/ns,1/ni]	An FLQuant object
largs*		list	1	A list with extra optional arguments:
	stk.gB	numeric	1	Surplus production (only for stocks in biomass)

na: number of age (from min.age to max.age)
ny(hist): number of historic years (from first.yr to proj.yr-1)
1/nu(stock): 1 or number of units of the stock
1/ns: 1 or number of seasons
1/ni: 1 or number of iterations